НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій Стиренко

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп,ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Розробка модулю налаштування программи обробки заказів колл-центру»

Виконав (-ла):
студент (-ка) IV курсу, групи ІП-64
Маркушевський Роман Сергійович _____

Керівник:
старший викладач
Андрій СІМОНЕНКО _____

Консультант з нормоконтролю:
доктор технічних наук, професор
Валерій СІМОНЕНКО _____

Рецензент:

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Бакалавр»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій Стіренко

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Маркушевському Роману Сергійовичу

1. Тема проєкту «Розробка модулю налаштування программи обробки заказів колл центру», керівник проєкту старший викладач Симоненко А.В., затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом проєкту 6.06.2020

3. Вихідні дані до проєкту

текст, html, дані про закази, дані користувачів.

4. Зміст пояснювальної записки

Опис завдання; опис предметної області і напрямів дослідження; аналіз і характеристика об’єкта проектування; обґрунтування оптимального варіанта реалізації мети бакалаврської роботи; опис алгоритму і програмного забезпечення; структура проектування системи і її компонентів; основні рішення з реалізації системи в цілому і її компонентів; опис використовуваного системного програмного забезпечення; інструкція роботи користувача з системою.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	Сімоненко В.П.	15.12.2019	03.02.2020
Розділ 2	Сімоненко В.П.	08.02.2020	11.03.2020
Розділ 3	Сімоненко В.П.	11.03.2020	09.04.2020
Розділ 4	Сімоненко В.П.	09.04.2020	30.04.2020

7. Дата видачі завдання 09.12.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми роботи*	09.12.2019	
2.	*Вивчення та аналіз завдання*	15.12.2019 – 03.02.2020	
3.	*Розробка архітектури та загальної структури систем*	08.02.2020 – 11.03.2020	
4.	*Розробка структур окремих підсистем*	11.03.2020 – 09.04.2020	
5.	*Програмна реалізація системи*	09.04.2020 – 30.04.2020	
6.	*Оформлення пояснювальної записки*	05.05.2020 – 10.05.2020	
7.	*Захист програмного продукту*		
8.	*Передзахист*		
9.	*Захист*		

Студент Маркушевський Р.С.

Керівник Сімоненко А.В.

Анотація

В бакалаврській *дипломній* роботі реалізований модуль для налаштування системи обробки заявок колл-центру.

Модуль дає можливість налаштовувати смс-повідомлення клієнтам, моніторити роботу операторів, проводити налаштування робочого процессу користувачів CRM. Продукт створений на мові програмування Java у середовищі розробки IntelliJ IDEA, з використанням веб-контейнеру сервлетів Apache Tomcat EE та базою даних PostgreSQL. Для візуалізації проекту використовуються інтернет-браузери.

Аннотация

В бакалаврской дипломной работе реализован модуль для настройки системы обработки заявок колл-центра.

Модуль позволяет настраивать смс-сообщение клиентам, мониторить работу операторов, производить настройку рабочего процесса пользователей CRM. Продукт создан на языке программирования Java в среде разработки IntelliJ IDEA, с использованием веб-контейнера сервлетов Apache Tomcat EE и базой данных PostgreSQL. Для визуализации проекта используются интернет-браузеры.

Annotation

In the bachelor's thesis the module for setting up the call center application processing system is implemented.

The module allows you to configure SMS messages to customers, monitor the work of operators, configure the workflow of CRM users. The product is created in the Java programming language in the IntelliJ IDEA development environment, using the Apache Tomcat EE servlet web container and the PostgreSQL database. Internet browsers are used to visualize the project.

Справки	Формат	*Позначення*	*Найменування*	Кількість листів	екземпляра№	*Примітка*
			Документация общая			
			Вновь разработанная			
	А4	***ИАЛЦ.466538.002 ТЗ***	Система защиты информации	1		
			в беспроводных сетях			
			Техническое задание			
	А4	***ИАЛЦ.466538.003 ТП***	Система защиты информации	1		
			в беспроводных сетях			
			Ведомость технического			
			Проекта			
	А4	***ИАЛЦ.466538.004 ПЗ***	Система защиты информации	61		
			в беспроводных сетях			
			Пояснительная записка			
	А1	***ИАЛЦ.466538.005 Д1***	Система защиты информации	1		
			в беспроводных сетях			
			Блок-схема центра			
	А1	***ИАЛЦ.466538.006 Д2***	Система защиты информации	1		
			в беспроводных сетях			
			Алгоритмы защиты			
			информации А3,А5,А8			
	А1	***ИАЛЦ.466538.007 Д3***	Система защиты информации	1		
			в беспроводных сетях			
			Метод защиты информации			
			в сетях			
		Дискета 3,5»		1		

					ИАЛЦ.466538.001 ОА
Змн	Арк.	№ докум.	Підпис	Дата	
Розроб.		Маркушевський Р. С.			Розробка модулю налаштування программи обробки заказів колл центру. Опис альбому
Перевір.		Сімоненко А.В.			
Н. Контр.		Сімоненко В.П.			
Затверд.		Стіренко С.Г.			

Літ.			Арк.	Аркушів

Технічне завдання

до дипломного проекту

на тему: Розробка модулю нашалтування системи обробки заявок колл-центу

Київ - 2020

Зміст

					?ИАЛЦ.467200.002 ТЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Маркушевський			Розробка модулю налаштування ПЗ для обробки замовлень колл центром. Технічне завдання.	Літ.	Арк. Аркушів
Перевір.		Сімоненко А.В.					
Реценз.						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, ІП-64	
Н. Контр.		Сімоненко В.П.					
Затверд.		Стіренко С.Г					

1.Найменування та область застосування

Найменування: «Розробка модулю нашалтування системи обробки заявок колл-центу».

Область застосування: CRM-система WhoCPA.

2.Підстави для розробки

Підставою для розробки системи обробки замовлень в Інтернет магазині є завдання на дипломне проектування, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (НТУУ «КПІ ім. І. Сікорського»)

3.Мета та призначення розробки

Метою даної розробки є розробка модулю нашалтування системи обробки заявок колл-центу.

Призначення даної розробки полягає в створенні програмного забезпечення яке надає можливість відстежувати стан замовлень,обробляти замовлення, моніторинг роботи операторів, .

4.Джерела розробки

Джерелом розробки є науково-технічна література по інформаційним питанням та публікації в глобальній мережі Інтернет з даних питань.

					ИАЛЦ.467200.002 ТЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Маркушевський			Розробка модулю налаштування ПЗ для обробки замовлень колл центром. Технічне завдання.	Літ.	Арк. / Аркушів
Перевір.		Сімоненко А.В.					
Реценз.						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, ІП-64	
Н. Контр.		Сімоненко В.П.					
Затверд.		Стіренко С.Г					

Пояснювальна записка
до дипломного проєкту
на тему: «Розробка модулю налаштування программи обробки заказів колл-центру»

Київ – 2020 року

Зміст

ВСТУП

Управління відносинами з клієнтами (англ. Customer relationship management (CRM), укр. Сі-ар-ем) — це поняття, яке охоплює концепції, котрі використовуються для управління взаємовідносинами з клієнтами, включаючи збір, зберігання й аналіз інформації про клієнтів, постачальників, партнерів та інформації про відносини з ними.

CRM - модель взаємодії, заснована на теорії, що центром всієї філософії бізнесу є клієнт, а головними напрямками діяльності компанії є заходи щодо забезпечення ефективного маркетингу, продажів і обслуговування клієнтів. Підтримка цих бізнес-цілей включає збір, зберігання і аналіз інформації про споживачів, постачальників, партнерів, а також про внутрішні процеси компанії. Функції для підтримки цих бізнес-цілей включають продажу, маркетинг, підтримку споживачів.

Система управління взаємовідносинами з клієнтами (CRM, CRM-система, скорочення від англ. Customer Relationship Management) - прикладне програмне забезпечення для організацій, призначене для автоматизації стратегій взаємодії з замовниками (клієнтами), зокрема для підвищення рівня продажів, оптимізації маркетингу і поліпшення обслуговування клієнтів шляхом збереження інформації про клієнтів і історію взаємин з ними, встановлення і поліпшення бізнес-процесів і подальшого аналізу результатів.

Сучасна CRM полягає у вивченні ринку і конкретних потреб клієнтів. На основі цих знань додаються нові товари або послуги і таким чином компанія досягає успіху упоставлених цілях і покращує свої фінансові показники.

Основні принципи:

Наявність єдиного сховища інформації, куди збираються відомості про взаємодію з клієнтами - клієнтської бази.

Використання багатьох каналів взаємодії: обслуговування на точках продажів, телефонні дзвінки, електронна пошта, заходи, зустрічі, реєстраційні форми на веб-сайтах, рекламні посилання, чати, соціальні мережі.

Існує три CRM-підходи, кожен з яких може бути реалізованим окремо від інших:

Оперативний — автоматизація споживчих бізнес-процесів, що допомагає персоналу з роботи з клієнтами виконувати свої функції.

Співробітницький — програма взаємодіє зі споживачами без участі персоналу з роботи з клієнтами.

Аналітичний — аналіз інформації про споживачів із різноманітними цілями.

CRM-система може включати:

фронтальну частину, що забезпечує обслуговування клієнтів на точках продажів з автономної, розподіленої або централізованої обробкою інформації;

операційну частину, що забезпечує авторизацію операцій і оперативну звітність;

сховище даних;

аналітичну підсистему;

розподілену систему підтримки продажів: репліки даних на точках продажів або смарт-карти.

Класифікації CRM-систем

Класифікація за призначенням

Автоматизована система управління продажами (англ. Sales force automation; SFA)

управління маркетингом

Управління клієнтським обслуговуванням і колл-центрами (системи з опрацювання звернень абонентів, фіксація і подальша робота зі зверненнями клієнтів)

Класифікація за рівнем обробки інформації

Операційний CRM - реєстрація та оперативний доступ до первинної інформації щодо подій, компаніям, проектам, контактам.

Аналітичний CRM - звітність і аналіз інформації в різних розрізах (воронка продажів, аналіз результатів маркетингових заходів, аналіз ефективності продажів в

розрізі продуктів, сегментів клієнтів, регіонів і інші можливі варіанти).

Коллаборатівний CRM (англ. Collaboration - співпраця; спільні, узгоджені дії) - рівень організації тісної взаємодії з кінцевими споживачами, клієнтами, аж до впливу клієнта на внутрішні процеси компанії (опитування, для зміни якостей продукту або порядку обслуговування, веб-сторінки для відстеження клієнтами стану замовлення, повідомлення по SMS про події, пов'язані із замовленням або лицьовим рахунком, можливість для клієнта самостійно вибрати і замовити в режимі реального часу продукти і послуги, а також інші інтерактивні можливості).

Ринок CRM-систем аналітиками Gartner за результатами 2012 року оцінений в обсязі $ 18 млрд, найбільші виробники - Salesforce.com (14%), SAP (12,9%) і Oracle (11,1%). Серед CRM-систем, що надаються за моделлю SaaS, більш ніж половиною ринку володіє Salesforce.com, серед інших помітних гравців в цьому сегменті виділяють також NetSuite і RightNow (поглинена Oracle в 2011 році).

Метою роботи є створення модулю для налаштування системи взаємодії з клієнтами. Модуль повинен містити в собі:

1. Перегляд статистики по споживачам та по замовленням.
2. Налаштування продуктів.
3. Створення аккаунтів для операторів колл-центу.
4. Об’єднання операторів у групи, в залежності від товару.
5. Прив’язування товарів до груп операторів.
6. Розподілення товарів між постачальниками.

(https://ru.wikipedia.org/wiki/
Система_управления_взаимоотношениями_с_клиентами)

РОЗДІЛ 1.

Досвід розробки систем взаємодії з клієнтами.

Впровадження CRM-стратегії в сучасному бізнесі неможливо без використання CRM-інструменту. CRM-система дозволяє акумулювати і аналізувати дані по клієнтах, розробляти оптимальні умови співпраці і в оперативному режимі відстежувати стан взаємодії з клієнтами, реагувати на їхні потреби.

Сучасна CRM-система, на думку відомого світового гуру в області CRM Бартона Голденберг, повинна містити наступні функціональні блоки:

управління контактами (і клієнтською базою);

управління продажами;

продажу по телефону (телемаркетинг);

управління часом (тайм-менеджмент);

підтримка і обслуговування клієнтів;

управління маркетингом (в т.ч. анкетуванням, опитуваннями, розсилками);

звітність для вищого керівництва;

інтеграція з іншими системами;

синхронізація даних;

управління електронною торгівлею (інтеграція з сайтом компанії, портал для клієнтів або партнерів);

управління мобільними продажами (з КПК, ноутбука або віддалений доступ).

Взаємодія з клієнтами тісно пов'язане з оперативною діяльністю з продажу товарів і послуг, і тому підтримується всіма типовими рішеннями системи "1С: Підприємство 8" для автоматизації торговельної діяльності:

"Управління торгівлей"

"BAS Роздрібна торгівля"

Управління торговим підприємством.

Для поглибленого забезпечення взаємодії з клієнтами в процесі продажів

призначена спеціалізована лінійка рішень "CRM":

CRM ПРОФ,

CRM Стандарт,

CRM КОРП,

Продукти цієї лінійки містять в собі такі можливості:

Ведення клієнтської бази

Управління контактами з клієнтами

Управління процесом продажу

Збір і зберігання інформації про клієнтів

аналіз клієнтів

Планування і аналіз ефективності роботи менеджерів з продажу

(https://tqm.com.ua/likbez/crm/crm-protsessy-v-konfiguratsiyah-1c-8#klienty)

Ведення клієнтської бази

"CRM" дозволяє створити клієнтську базу необхідної структури і розподілити клієнтів між сотнуднікамі. Для структури клієнтської бази використовуються прості і експертні аналітики, а також види контактної інформації, необхідні компанії для взаємодії з клієнтами. Відповідно до регламентами аналітик і контактної інформації настроюються звіти, які контролюють повноту заповнення клієнтської бази. Для керівництва і служби маркетингу в будь-який момент доступна інформація по динаміці зростання клієнтської бази, кількісного та якісного розподілу клієнтів по менеджерам, ємності клієнтських сегментів.

1.1 Управління контактами з клієнтами

"CRM" дозволяє реєструвати і планувати телефонні переговори, особисті зустрічі, поштові, електронні листи та інші контакти з клієнтами. У рішенні організовано:

оперативна і зручна робота по поточних і прострочених контактам;

планування контактів на майбутнє з можливістю перегляду і редагування в індивідуальному календарі користувача;

створення нагадувань про майбутні контакти на певний час;

призначення відповідального за виконання контакту з повідомленням відповідального про необхідність контакту, його цілі і завдання;

передача інформації про контакти з клієнтом між співробітниками;

персоналізовані поштові та електронні розсилки групам клієнтів.

Керівництву в будь-який час доступні звіти про поточні, прострочених, запланованих контактах по клієнтах в розрізі менеджерів, про відсутність взаємодії з клієнтами протягом певного періоду.

Управління процесом продажу

В області управління продажами лінійки рішень "CRM" дозволяє:

організувати реєстрацію, обробку інтересів нових і діючих клієнтів до продукції і послуг компанії;

направляти керівнику оповіщення у вигляді електронного листа або sms-повідомлення при виникненні ситуації, що вимагає його безпосередньої участі;

аналізувати потенціал і перспективність клієнта, врахувати ризики можливої угоди;

надавати керівництву аналітику за кількістю і часу обробки нових інтересів і потреб клієнтів;

моделювати бізнес-процес продажу під специфіку компанії;

аналізувати стан процесу продажу за допомогою звіту "Воронка продажів". Порівнювати поточні показники етапів продажу зі статистикою і прийнятими в компанії стандартами;

готувати комерційні пропозиції з використанням прийнятих в компанії шаблонів, вести історію зміни комерційних пропозицій. Проводити аналіз пропозицій, підготовлених і відправлених конкретними співробітниками;

формувати і відправляти рахунки клієнтам компанії. Вести оперативну роботу з інформацією по оплаті, відвантаження. Контролювати прострочення оплати і відвантаження. (http://www.voit.pro/blog/istoriya-razvitiya-crm/)

аналізувати продажу, отримувати -ABC і -XYZ аналіз продажів; отримувати рейтинги найуспішніших менеджерів за допомогою ABC-аналізу;

аналізувати роботу менеджерів з продажу більш ніж по 25 різними показниками.

Основним завданням роздрібних продажів відносно роботи з покупцями є надання різних сервісів і програм, спрямованих на зручність обслуговування і отримання додаткових переваг при здійсненні покупки в одному магазині (або однієї мережі).

"BAS Роздрібна торгівля" є типовим рішенням і містить широкий спектр можливостей, спрямованих на підтримку роздрібних продажів.

Для аналізу покупців в програмі існують такі можливості, як:

інструменти для накопичення та аналізу відомостей про покупців, необхідних для розробки цільових маркетингових програм (стать, вік, кількість дітей, купівельні переваги і т.п.)

можливість реєстрації даних про покупця при видачі дисконтної (клубної) карти і періодичне оновлення цієї інформації при повторних відвідинах;

угруповання покупців за різними параметрами;

аналітика з продажу в розрізі груп покупців, яка дозволяє скорегувати асортимент магазину;

аналіз продажів в розрізі дисконтних карт (зареєстрованих покупців).

Для надання індивідуальних умов клієнтам продукт містить широкі можливості управління знижками до встановленої роздрібної ціни:

разові та накопичувальні знижки за сумою і за кількістю проданого товару;

за часом здійснення покупки;

в залежності від виду дисконтної карти покупця;

на підставі персоніфікованих подій покупця (день народження, 8 березня і т.п.);

знижки у вигляді відсотка від суми за товарною рядку;

подарунок при покупці набору товарів ("купи — отримай").

Прикладне рішення дозволяє надавати знижку при одночасному виконанні декількох різнорідних умов. Підтримуються накопичувальні порогові схеми, як з заміною, так і без заміни дисконтних карт покупця при перетині порогу, облік подарункових сертифікатів. Застосування знижки може здійснюватися автоматично або по команді співробітника магазина.

Для зручності розрахунків з покупцями і застосування знижок підтримується використання штріхкодірованних карт і карт із магнітною смугою.

"Управління торгівлею" є комплексним рішенням 1С: Підприємство, що підтримує цілий спектр взаємопов'язаних процесів торгового підприємства, у тому числі процесів взаємин з клієнтами.

У частині підтримки методології CRM програма пропонує наступні можливості:

Збір і зберігання інформації про клієнтів

Рішення забезпечує реєстрацію та систематизацію даних про існуючих і потенційних клієнтів (основні контактні дані, контактні особи та ін.), Ведення історії взаємин з клієнтом, реєстрацію планованих і контактів, що відбулися. Продукт дозволяє встановлювати зв'язки між декількома компаніями, об'єднаними діловими відносинами і аналізувати взаємодії з клієнтами з урахуванням наявних взаємозв'язків.

1.2 Аналіз клієнтів

У продукті підтримується ABC / XYZ-класифікація клієнтів, а також класифікація за станом відносин (потенційний клієнт, разовий клієнт, постійний клієнт, втрачений клієнт).

Планування і аналіз ефективності роботи менеджерів з продажу

Рішення забезпечує закріплення клієнтів за конкретними менеджерами, за допомогою бізнес-процесів дозволяє призначати відповідальних за певні етапи взаємодії з клієнтом і контролювати виконання завдань.

Для оцінки ефективності роботи співробітників відділу продажів передбачений звіт Порівняльний аналіз показників роботи менеджерів, який дає можливість

зіставляти результати за різними показниками ефективності, як між співробітниками, так і по співробітнику в динаміці за певний період. (https://tqm.com.ua/likbez/crm/crm-protsessy-v-konfiguratsiyah-1c-8#klienty)

ВИСНОВКИ ДО РОЗДІЛУ 1

Підводячи підсумки хочеться відзначити, один незаперечний факт - У сучасному діловому світі правила гри змінюються дуже динамічно. Кожні п'ять років у користувачів CRM відбувається нове сприйняття даного програмного продукту і з кожним роком CRM ставати все потрібніше і користується все більшим попитом. Чим більше компанія і вище її активність, тим більше у неї клієнтів і завдань - відповідно з'являється потреба в обробці великих клієнтських баз і великого обсягу даних, а це основне завдання CRM. Використання CRM збільшує обсяг продажів в рази, піднімає рівень доходу і прибуток організації.

РОЗДІЛ 2

ВИБІР МЕТОДУ І СТРАТЕГІЇ ВИРІШЕННЯ

Перш за все слід обрати мову програмування для вирішення проблеми. На сьогоднішній день існує багато мов програмування які є зручними для вирішення певних класів задач.

2.1 Обрання мови програмування

Мова ассемблера — низькорівнева мова програмування, спецефічна для процесора. Перевагою є висока швидкодія мови та невеликі ресурсні затрати для виконяння. Недоліком є специфічність для конкретної архітектури компьютера та набір лише дуже простих операцій. Такі мови добре підходять для програмування мікроконтролерів проте для більш масштабних систем затрати на налагодження коду є занадто великими.

C++ - мова програмування, що підтримує різні парадигми програмування та використовується у проектах різних типів. З одногу боку синтаксис C++ дозволяє швидше писати та налагоджувати код, а з іншого це мова з високою швидкодією. Проте C++ поруч з безпечними механізмами роботи з пам’яттю має і небезпечні, і неюезпечні писати легше і швидше ніж безпечні, що є мінусом. C++ використовують як для низькорівневого програмування, так і для більш високорівневих проектів. Проте доволі часто це старі проекти які просто підтримуються.

Java — мова програмування яка в основному націлена на об’єктно-орієнтований підхід і є мультиплатформеною. Перевагами є розвинутий інструмантарій для написання програм у об’єктно-орієнтованому стилі, переносимість вже скомпілбованої програми завдяки java-машині яка переводить звичайний java-код у універсальний байткод та велика кількість фреймворків, які спрощують як роботу з базою даних так і написання клієнт-серверного застосування.

Програми написані мовою програмування java повільніші за програми написані на c++, проте збільшується швидкість розробки та значно зменшена небезпечна робота з пам’яттю. В основному javaвикористовується для написання серверної

частини клієнт-серверних застосувань.

C# на платформі .Net — мудьтиплатформена та мультипарадигменна мова програмування. Перевагою платформи .Net є те що легко сумістити модулі написані на різних мовах програмування, що підтримуються даною платформою, оскільки всі вони претворюються у CIL(Common Intermidiate Language) який можна назвати аналогом байт-каду у java. Також реалізовані фреймворки як для зручної роботи з базою даних так і для написання веб-застосувань. Використовується як для написання серверної частини веб-застосуваня так і для ігрової розробки.

JavaScript — мова програмування з динамічною типізацією даних. Підходить як для написання серверної так і для написання клієнтської частини веб-застосування. Для написання серверної частини існує платформа node js, а для написання клієнткої частини такі фреймворки як angular та react. Гарно підтримує як об'єктно-орієнтований так і функціональний стилі програмування.

Python — мова програмування з динамічною типізацією та можливістю векторизаціїї. Використовується для проектів різних типів, від ботів до нейроних мереж. Завдяки векторизаціїї може виконуватися зі швидкістю, зрівненною з C++. Реалізовані бібліотеки для статистичної обробки даних, візуалізаціїї, кластеризаціїї/класифікації даних, створення нейроних мереж тощо.

Julia — мова програмування, яка має інструменти для створення розподілених програм.

Серед зазначених мов програмування було обрано мову програмування java окільки вона гарно підходить для розробки серверної частини веб-застосувань.

2.2 Обрання архітектури проекту.

MVC(Model View Controler) — архітектура застосування, у якій программа розділяється на 3 модулі: модель, у якій реалізовані бізнес-процеси, представлення, у якому реалізоване відображеня змін моделі для користувача та контролер, який контролює передачу даних між моделлю та продставленням. Модель та представлення не можуть обмінюватись даними напряму, тільки через контролер.

Данні змінює тільки модель, а представлення лише відображеє зміну користувачу.

MVP(Model View Presenter) — видозміна архітектури MVC. Тут модель — це просто представлення даних, а представлення — це лише відображення даних та отримання команд користувача. Основна бізнес-логіка знаходиться у презентері.

MVVC(Model View ViewModel) - видозміна архітектури MVC. Головною відмінністю є те, що посередник з одного боку є моделлю, представленою у вигляді представлення, а з іншого є абстракцією представлення. Така середня ланка дозволяє швидко обмінюватися даними між представленням і моделлю.

Багатошарова архітектура — архітектура яка розділяє застосування на 3 шари: DAL(Data Access Layer), BLL(Business Logic Layer), AL(Application Layer), PL (Presentation Layer). Тільки сусідні шари можуть обмінюватись даними. DAL відповідає за низькорівневу роботу з даними, в основному запис/читання з бази даних. У BLL реалізована бізнес логіка. У AL реалізовані функціїї контролера, а у PL представлення даних.

Для даного проекту була обрана архітектура MVC, окільки вона найкраще підходить під задачі проекту.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було проаналізовано усі можливі технології для розробки данного програмного доданку. Також було аргументовано вибір технологій для розробки програмного забезпечення.

РОЗДІЛ 3

ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОЕКТУ

3.1 Проектування програмного проекту.

Проектування програмного забезпечення - процес створення проекту програмного забезпечення (ПО), а також дисципліна, що вивчає методи проектування. Проектування ПО є окремим випадком проектування продуктів і процесів.

Метою проектування є визначення внутрішніх властивостей системи і деталізації її зовнішніх (видимих) властивостей на основі виданих замовником вимог до ПО (вихідні умови задачі). Ці вимоги піддаються аналізу.

Проектування ПО включає такі основні види діяльності:

вибір методу і стратегії вирішення;

вибір уявлення внутрішніх даних;

розробка основного алгоритму;

документування ПЗ;

тестування і підбір тестів;

вибір представлення вхідних даних.

Програмний проект повинен мати можливість проводити налаштування системи CRM. Додавати нових користувачів, прослуховувати записи дзвінків, виконувати функції оператора (обробляти замовлення)

(https://ru.wikipedia.org/wiki/Проектирование_программного_обеспечения)

Схема загальної архітектури проекту:

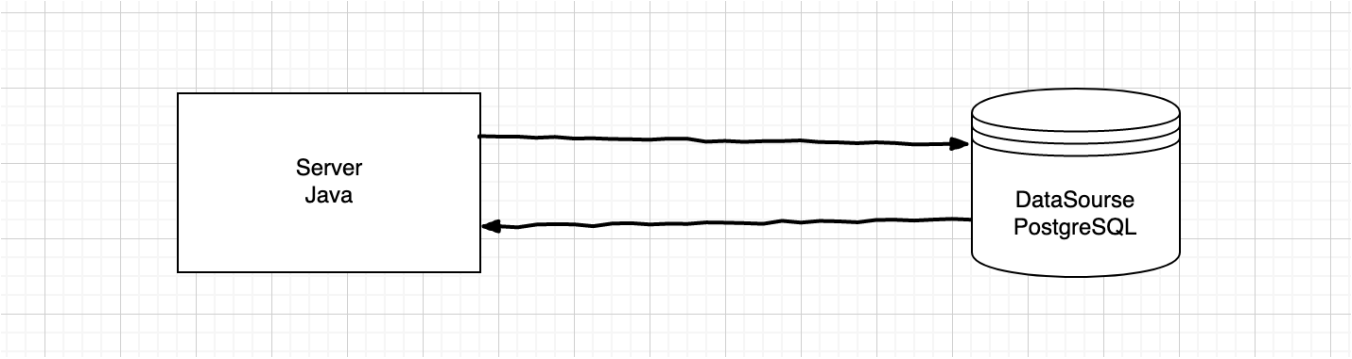


Рис. 3.1 - Схема загальної архітектури проекту

Аутентифікація користувача:

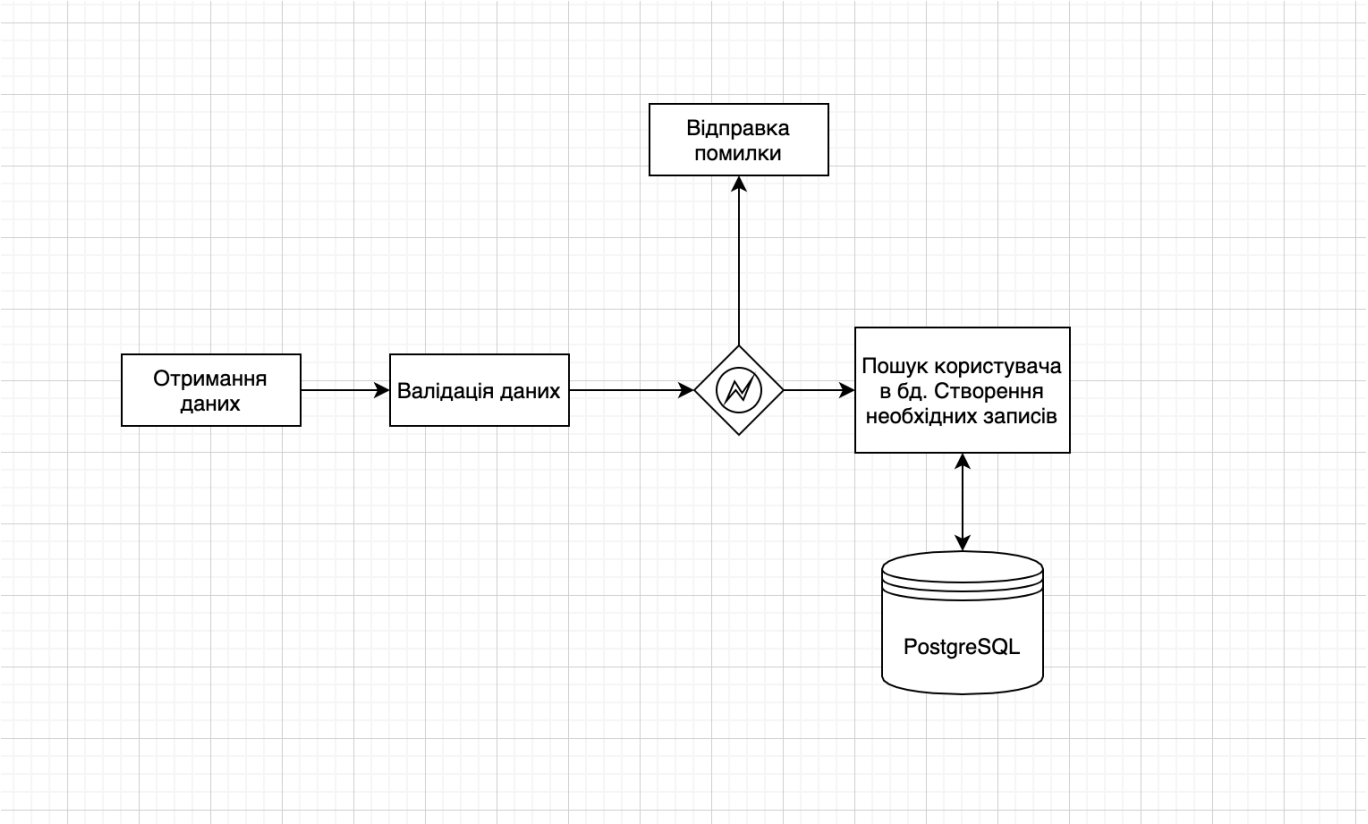


Рис. 3.2 - Аутентифікація користувача

Алгоритм:

1. Отримання даних для аутентифікації з клієнтської частини до серверної
2. Обробка даних на сервері
3. Якщо помилка — то повернення помилки
4. Інакше — переадресація на головну сторінку менеджера.

Отримання записів дзвінків:

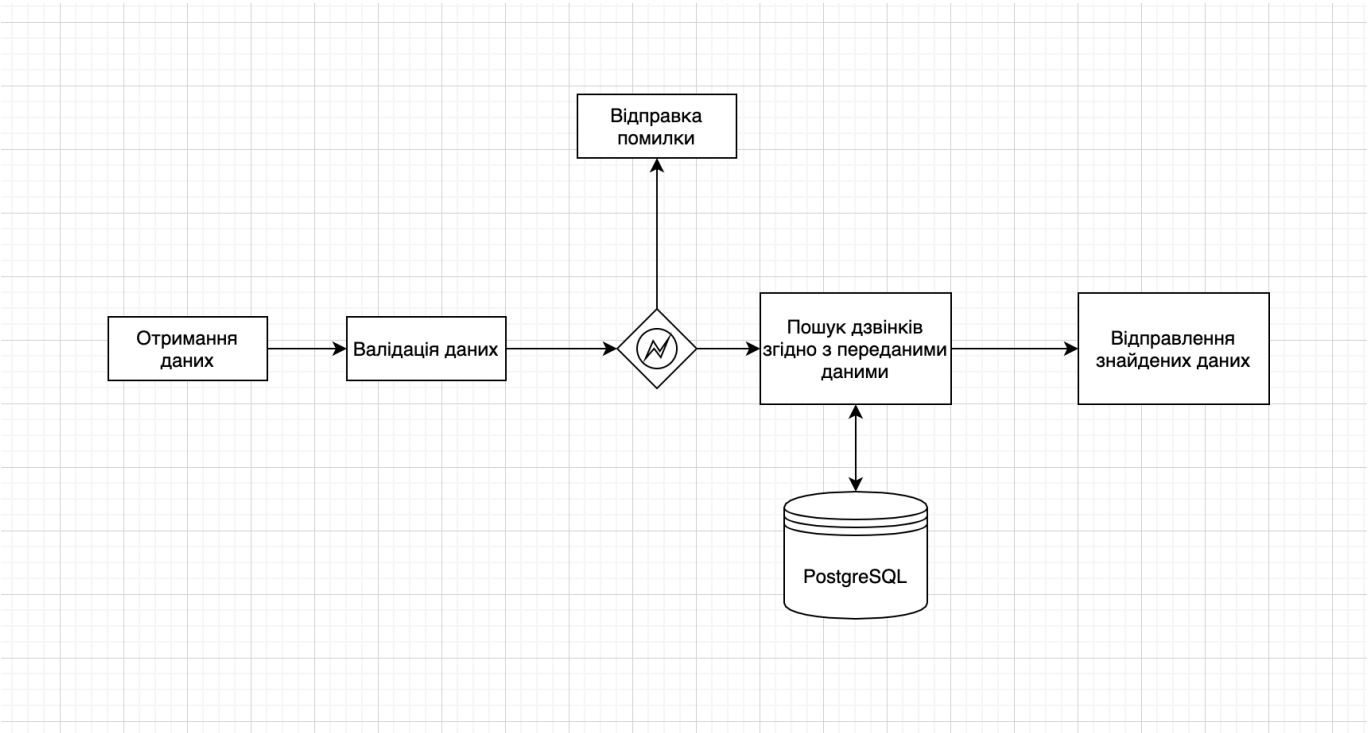


Рис. 3.3 - Отримання записів дзвінків

Алгоритм:

1. Отримання даних для отримання записів дзвінків
2. Обробка даних на сервері
3. Якщо помилка — то повернення помилки
4. Інакше — відкриття діалогового вікна з записами дзвінків.

Отримання списку замовлень:

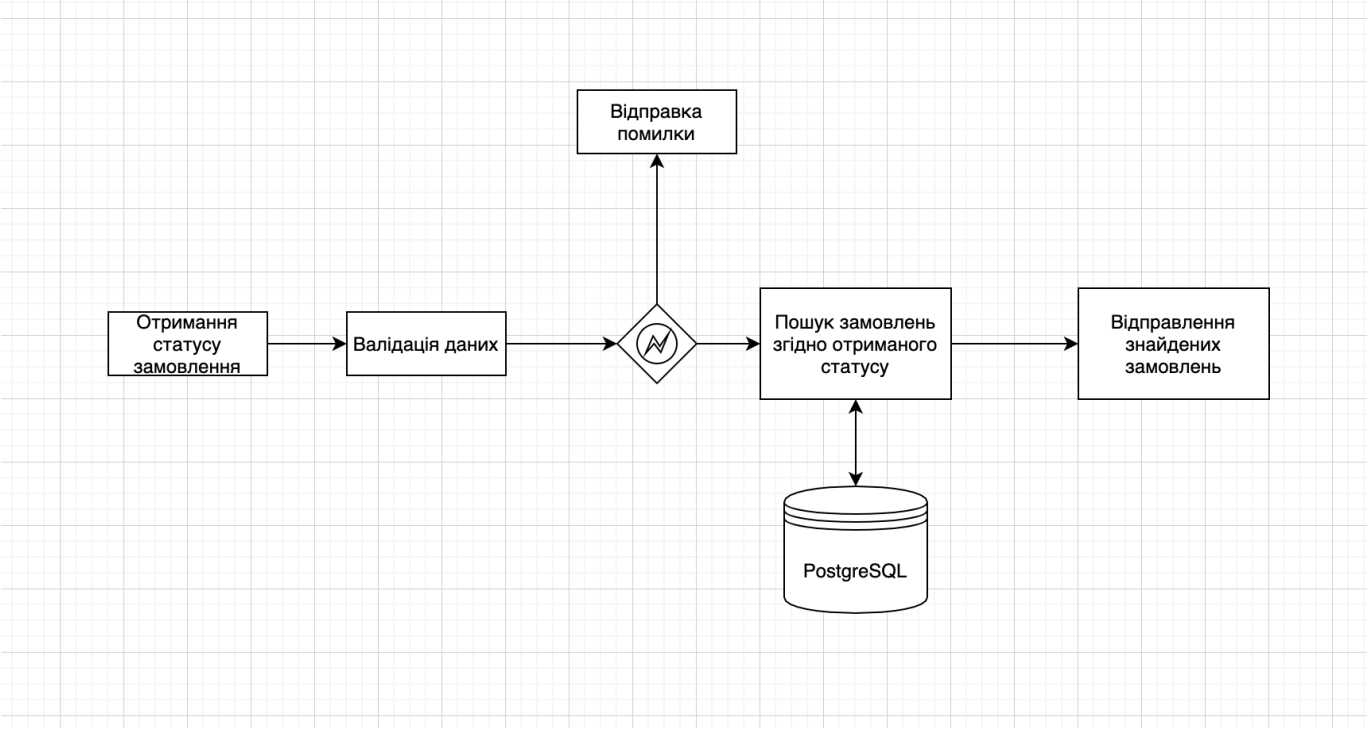


Рис. 3.4 - Отримання списку замовлень

Алгоритм:

1. Отримання статусу замовлення
2. Обробка даних на сервері
3. Якщо помилка — то повернення помилки
4. Інакше — відкриття діалогового вікна з замовленнями.

Створення нового замовлення:

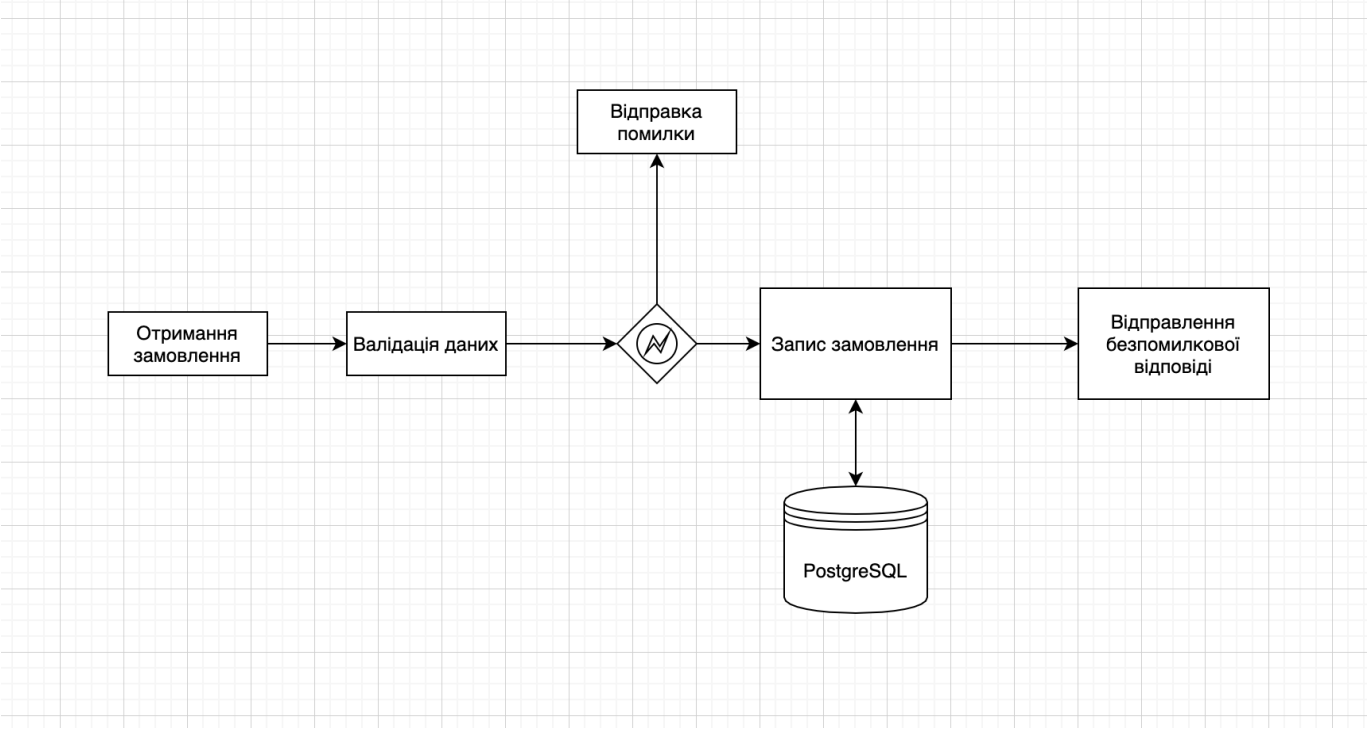


Рис. 3.5 - Створення нового замовлення

Алгоритм:

1. Отримання замовлення
2. Обробка даних на сервері
3. Якщо помилка — то повернення помилки
4. Інакше — відправлення відповіді 200.

Відкриття замовлення:

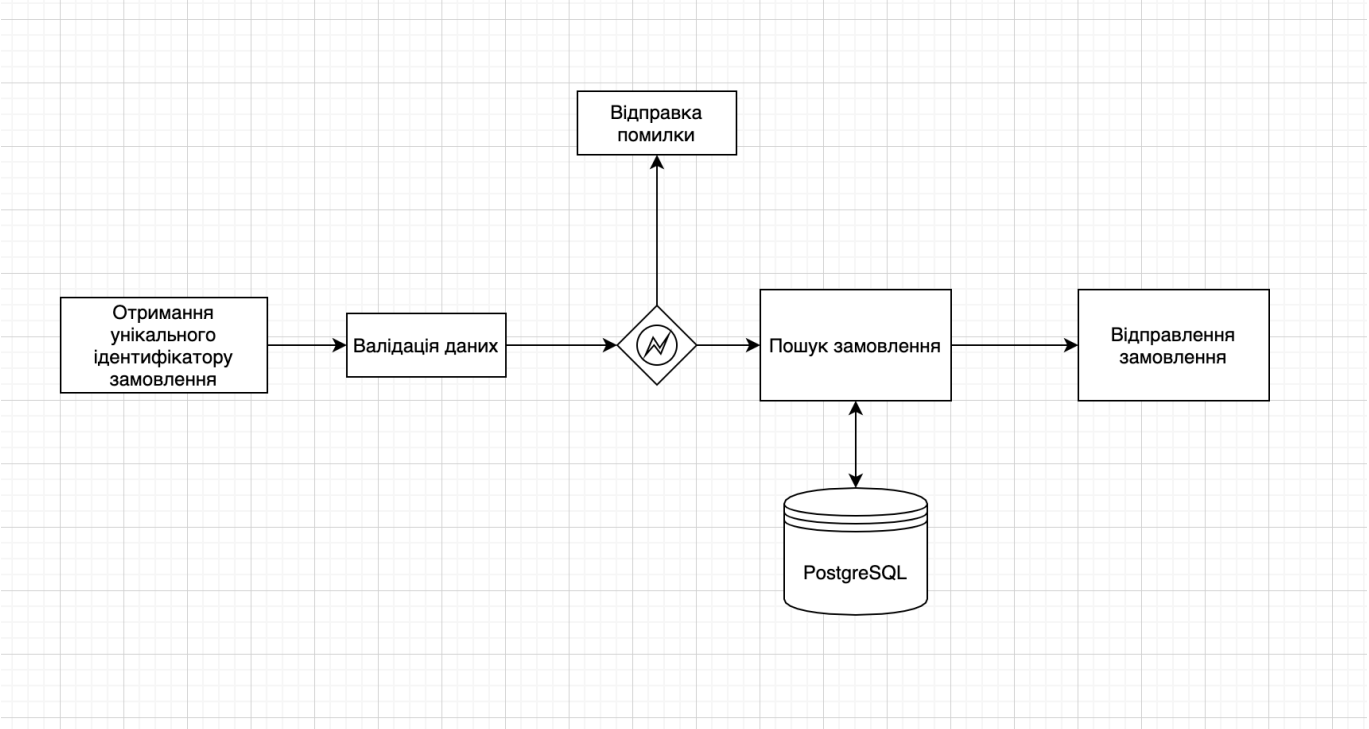


Рис. 3.6 - Відкриття замовлення

Алгоритм:

1. Отримання унікального ідентифікатору замовлення
2. Обробка даних на сервері
3. Пошук замовлення у базі.
4. Якщо помилка — то повернення помилки
5. Інакше — відправлення знайденного замовлення.

Редагування замовлення:

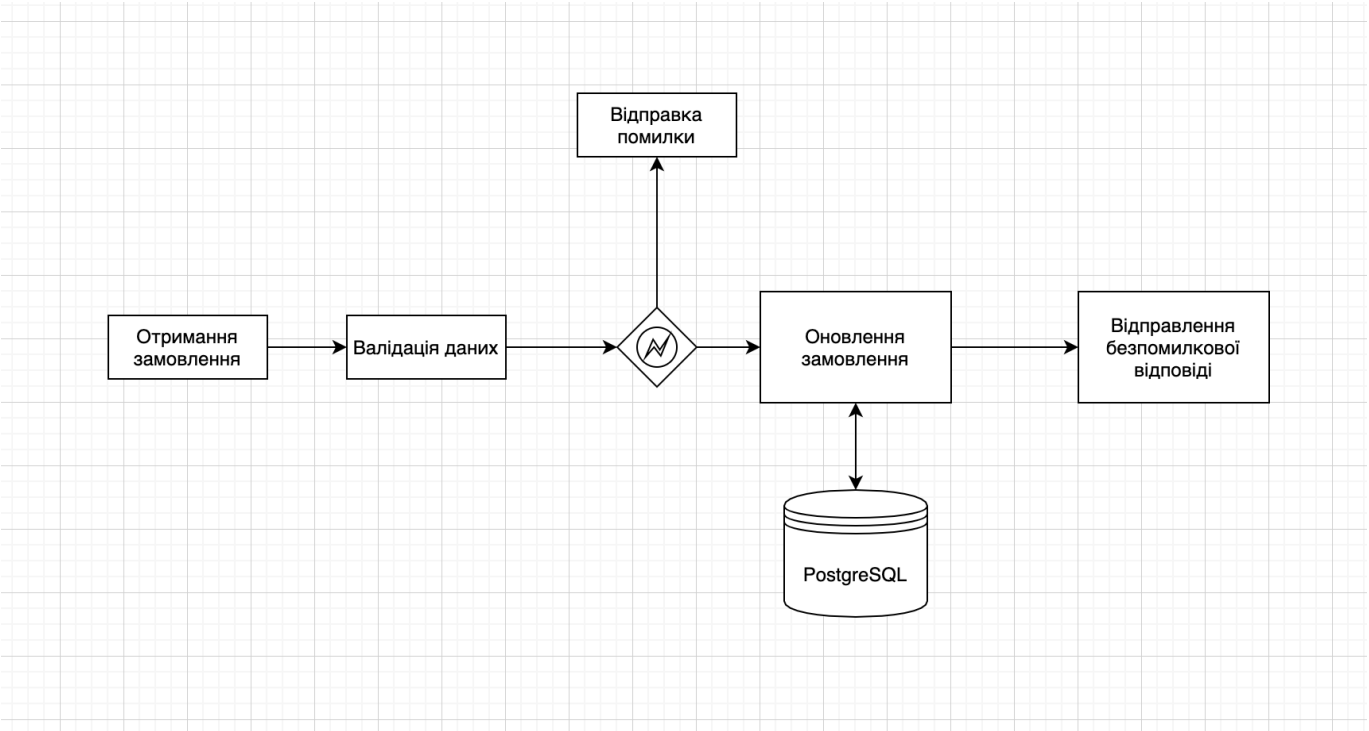


Рис. 3.7 - Редагування замовлення

Алгоритм:

1. Отримання замовлення
2. Обробка даних на сервері
3. Оновлення замовлення.
4. Якщо помилка — то повернення помилки
5. Інакше — відправлення 200.

Додавання шаблону для логування дій користувача:

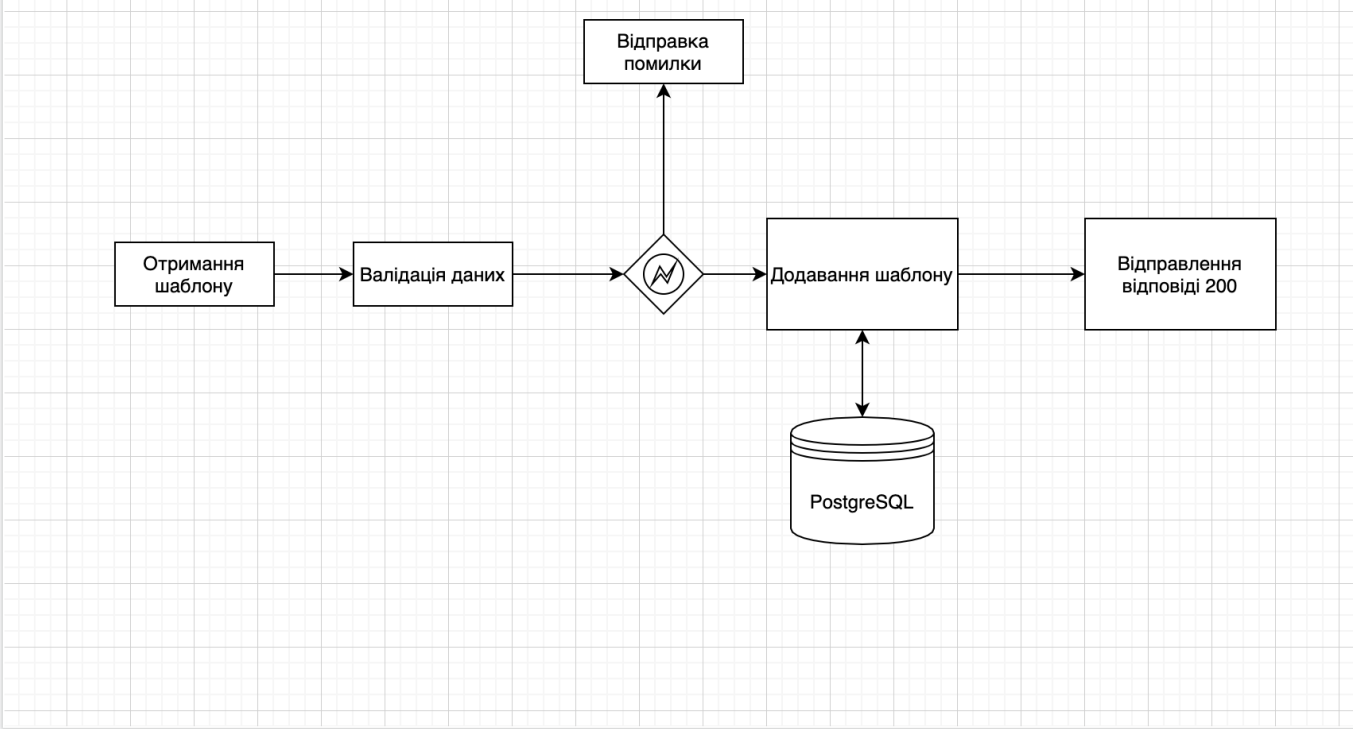


Рис. 3.8 - Додавання шаблону для логування дій користувача

Алгоритм:

1. Отримання шаблону
2. Обробка даних на сервері
3. Додавання шаблону.
4. Якщо помилка — то повернення помилки
5. Інакше — відправлення відповіді 200

Редагування шаблону для логування дій користувача:

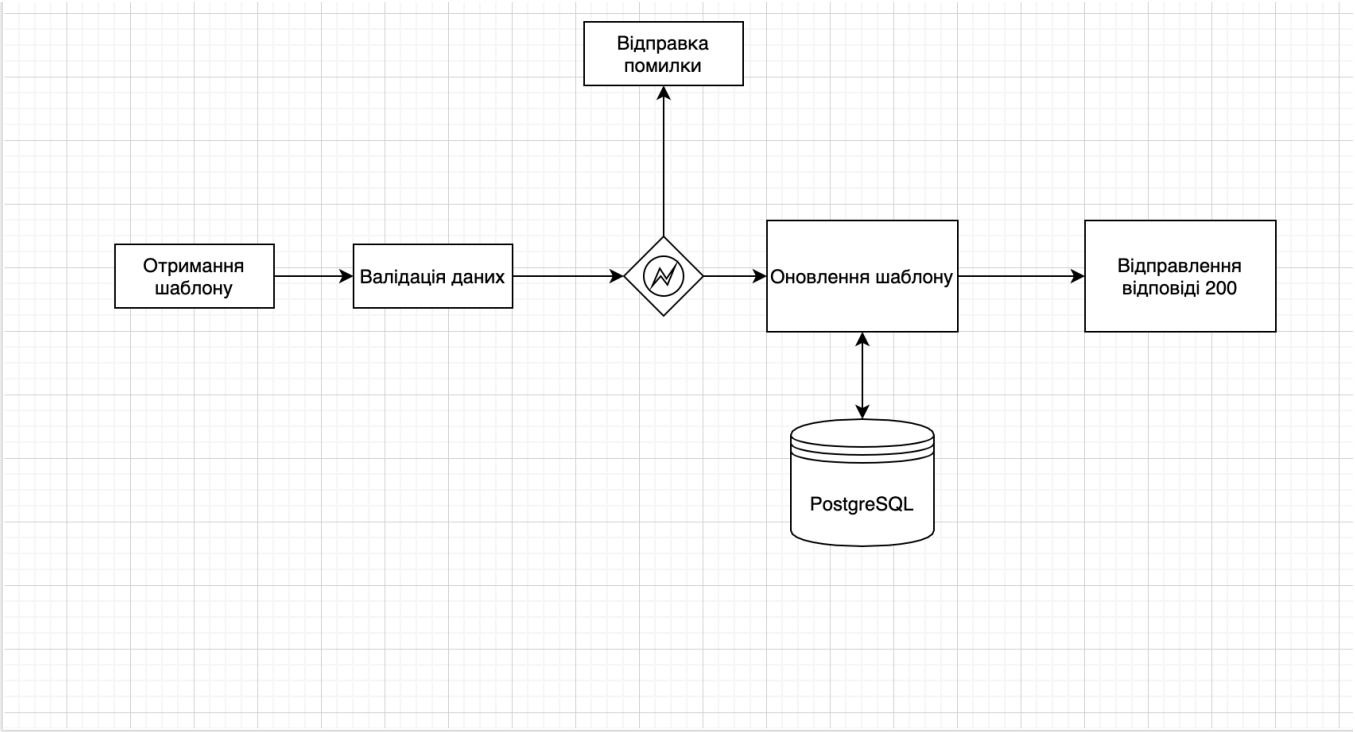


Рис. 3.9 - Редагування шаблону для логування дій користувача

Алгоритм:

1. Отримання шаблону
2. Обробка даних на сервері
3. Оновлення шаблону.
4. Якщо помилка — то повернення помилки
5. Інакше — відправлення відповіді 200

Отримання списку подій, які повинні логуватись:

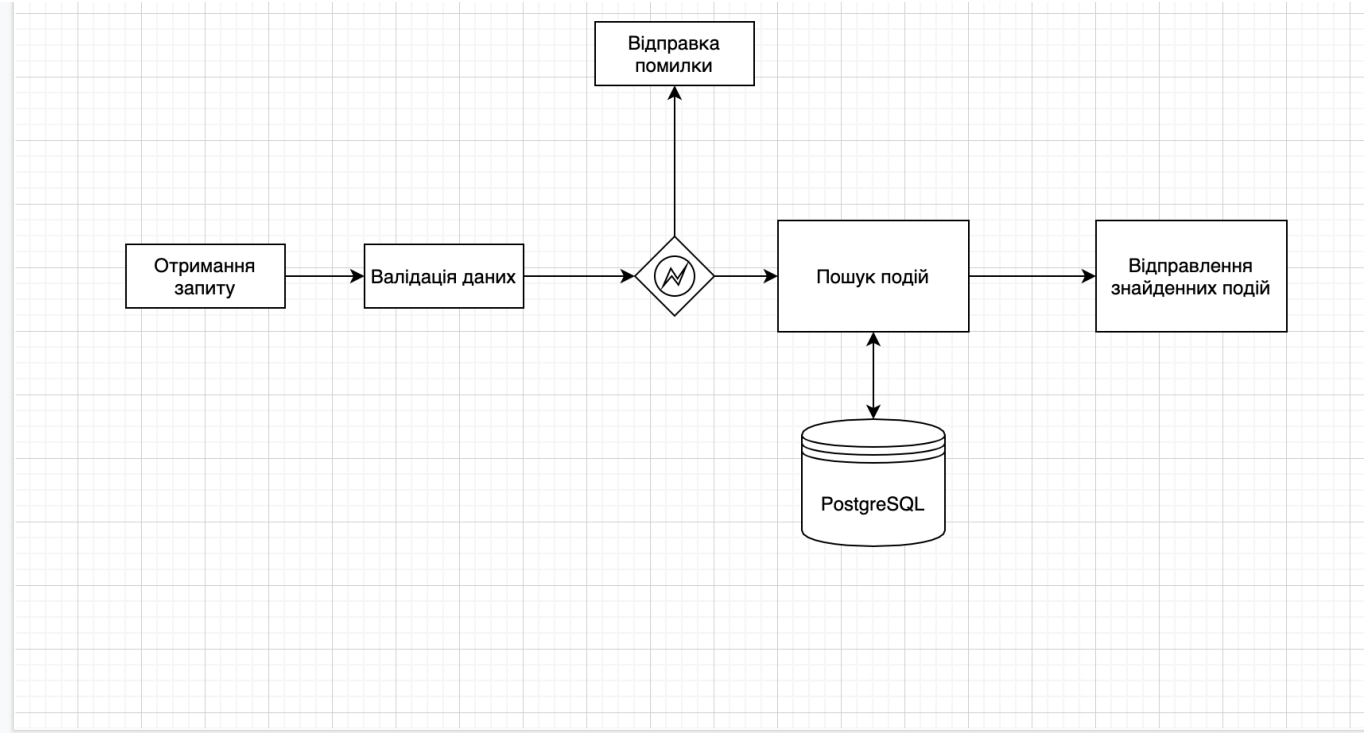


Рис. 3.10 - Отримання списку подій, які повинні логуватись

Алгоритм:

1. Отримання запиту
2. Обробка даних на сервері
3. Пошук подій.
4. Якщо помилка — то повернення помилки
5. Інакше — відправлення подій

Оновлення списку подій, які повинні логуватись:

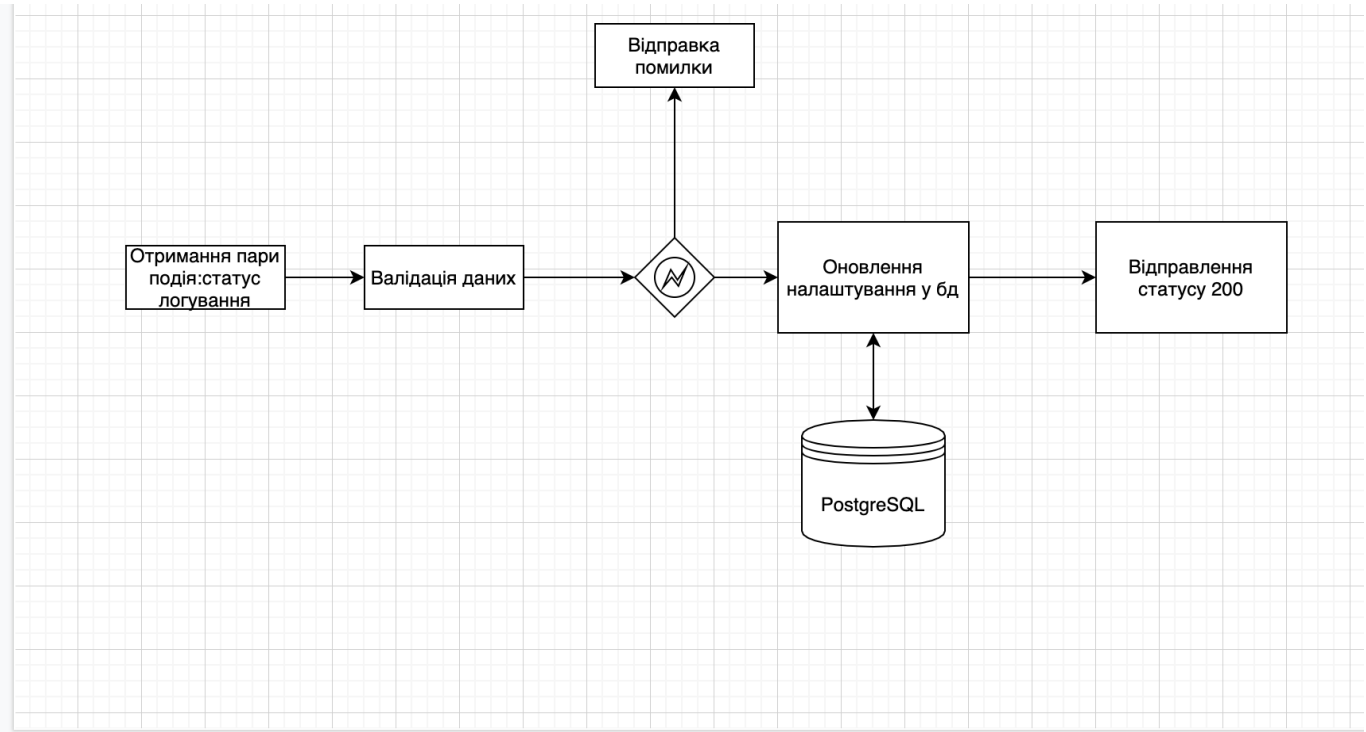


Рис. 3.11 - Оновлення списку подій, які повинні логуватись

Алгоритм:

1. Отримання пари подія:статус логування
2. Обробка даних на сервері
3. Оновлення налаштування у бд.
4. Якщо помилка — то повернення помилки
5. Інакше — відправлення 200

Отримання логу подій користувачів:

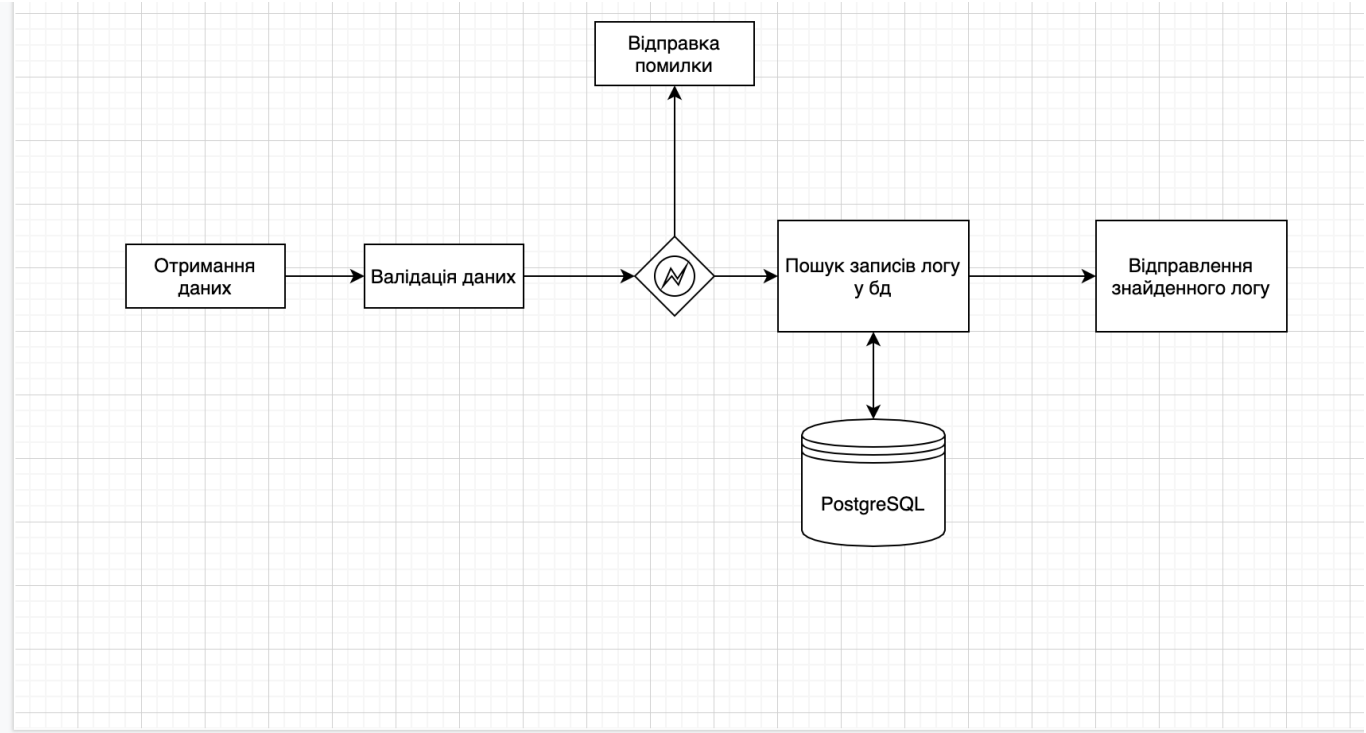


Рис. 3.12 - Отримання логу подій користувачів

Алгоритм:

1. Отримання даних.
2. Обробка даних на сервері.
3. Пошук логу у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення знайденного логу.

Отримання статистики по роботі операторів:

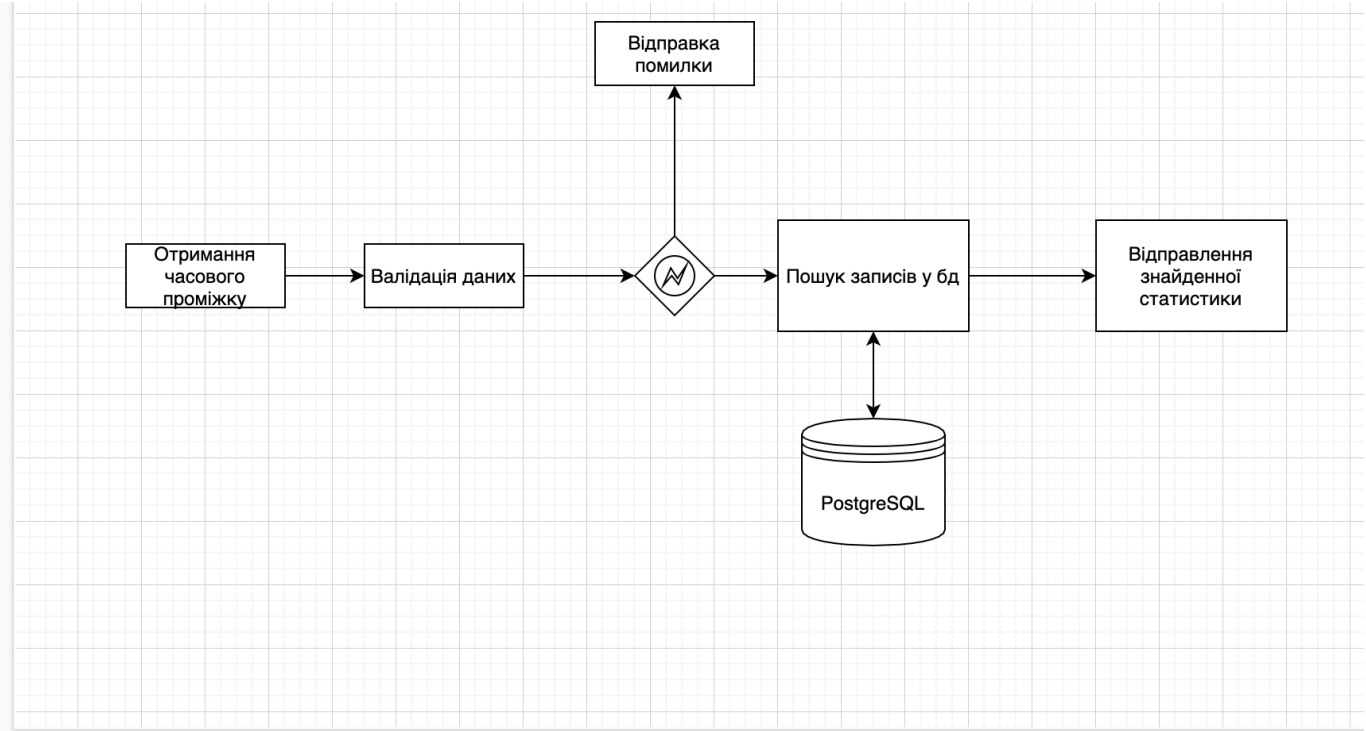


Рис. 3.13 - Отримання статистики по роботі операторів

Алгоритм:

1. Отримання часового проміжку.
2. Обробка даних на сервері.
3. Пошук статистики у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення знайденної статистики.

Отримання списку промо-акцій:

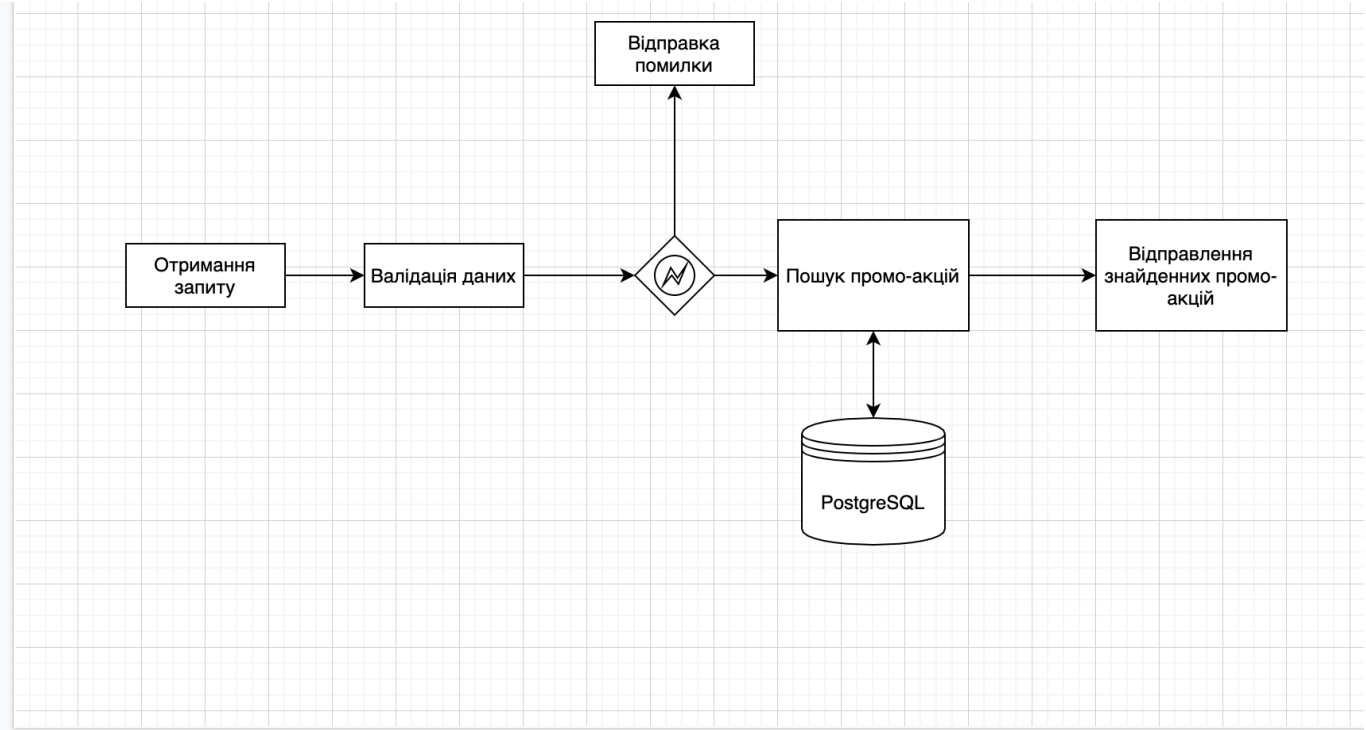


Рис. 3.14 - Отримання списку промо-акцій

Алгоритм:

1. Отримання запиту.
2. Обробка даних на сервері.
3. Пошук промо-акцій у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення знайденних промо-акцій.

Створення нової промо-акції:

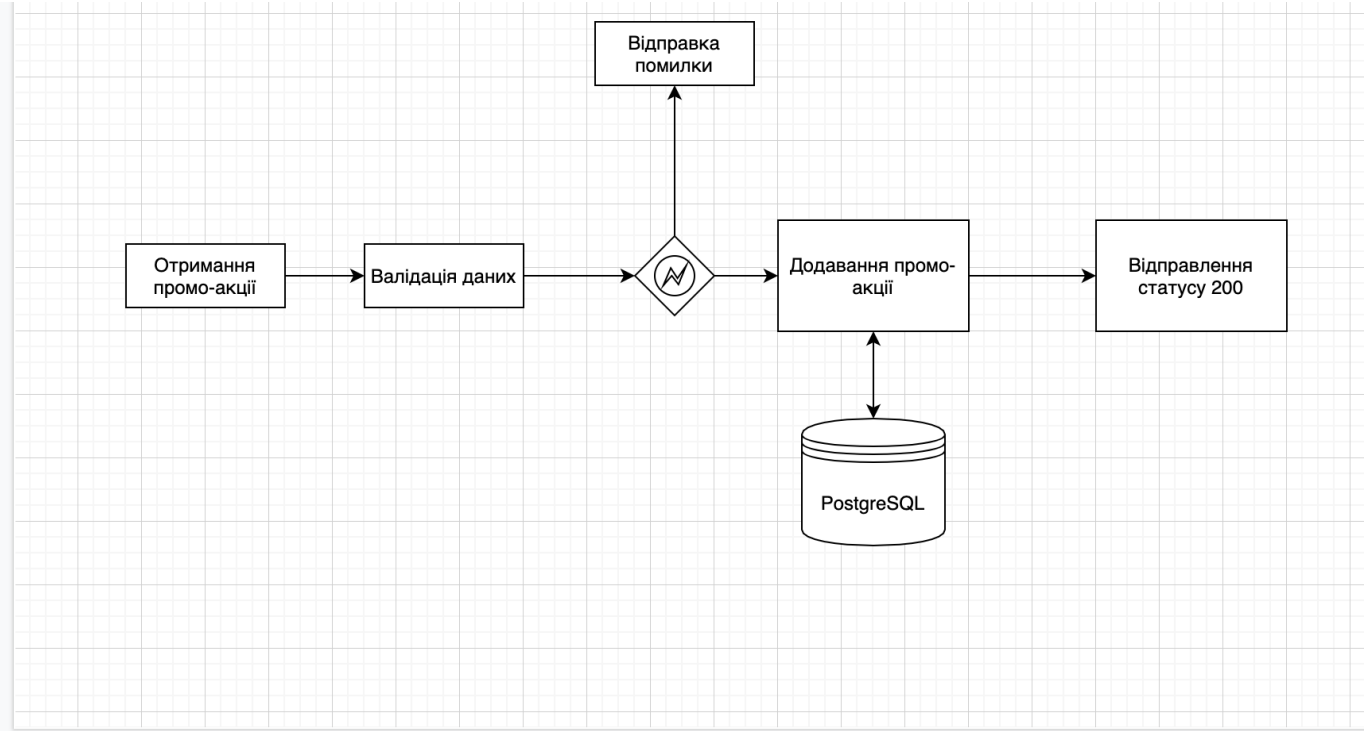


Рис. 3.15 - Створення нової промо-акції

Алгоритм:

1. Отримання промо-акції.
2. Обробка даних на сервері.
3. Додавання промо-акції у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення статусу 200.

Редагування промо-акції:

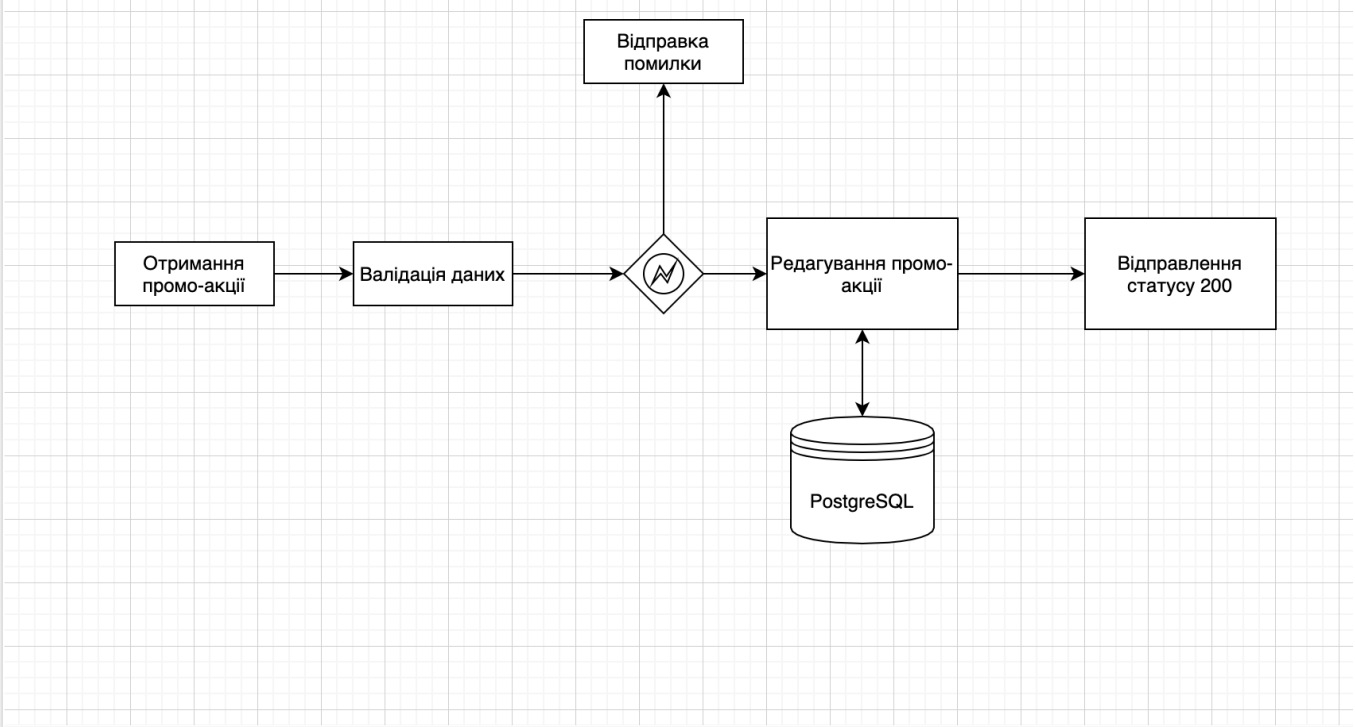


Рис. 3.16 - Редагування промо-акції

Алгоритм:

1. Отримання промо-акції.
2. Обробка даних на сервері.
3. Додавання промо-акції у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення статусу 200.

Отримання списку груп операторів:

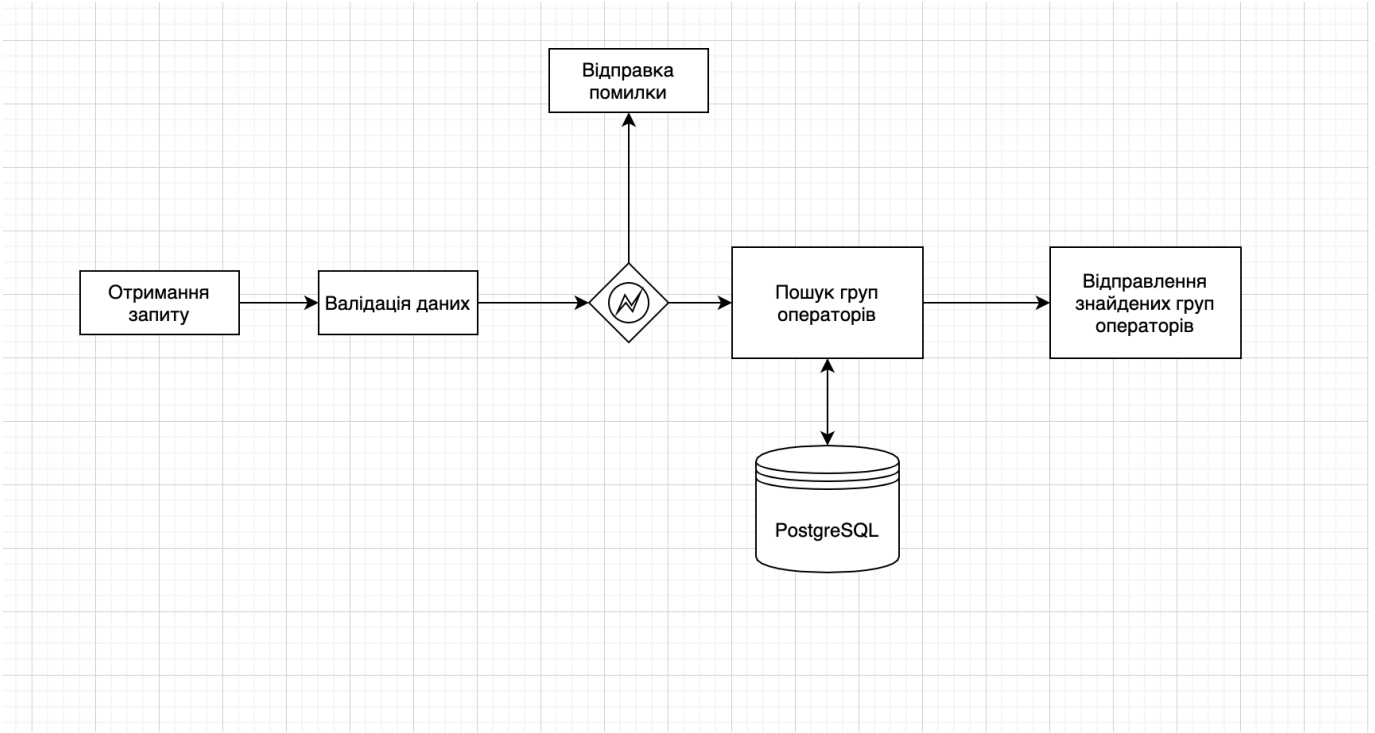


Рис. 3.17 - Отримання списку груп операторів

Алгоритм:

1. Отримання запиту.
2. Обробка даних на сервері.
3. Пошук груп операторів у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення знайдених груп операторів.

Додавання групи:

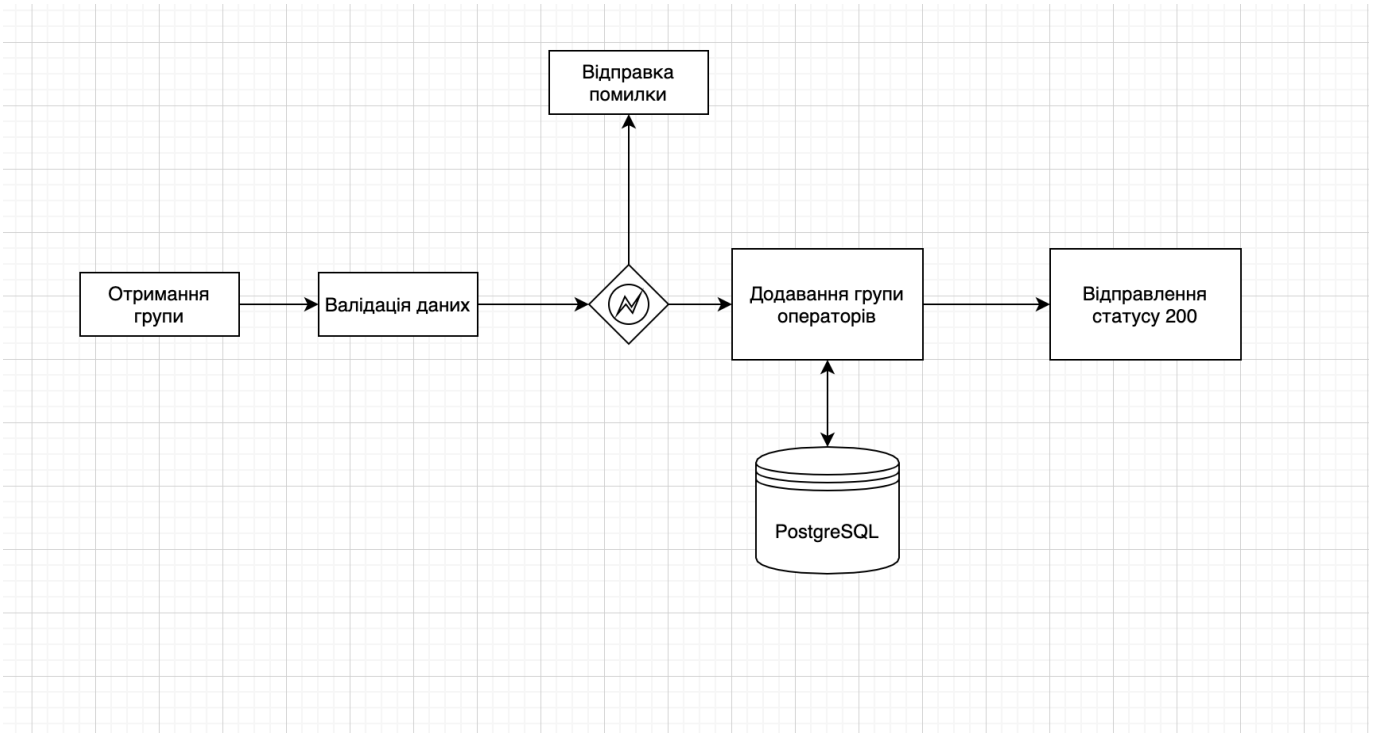


Рис. 3.18 - Додавання групи

Алгоритм:

1. Отримання групи.
2. Обробка даних на сервері.
3. Додавання групи операторів у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення статусу 200.

Редагування групи:

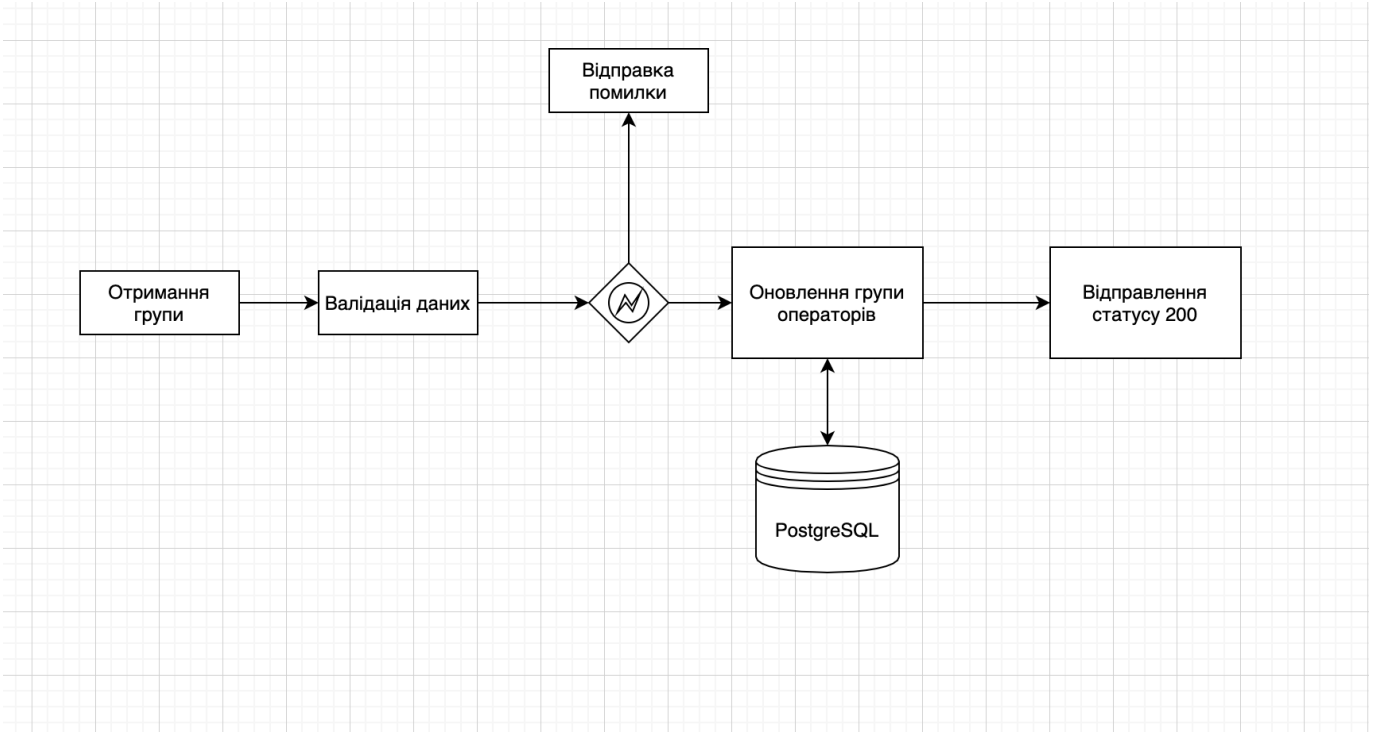


Рис. 3.19 - Редагування групи

Алгоритм:

1. Отримання групи.
2. Обробка даних на сервері.
3. Оновлення групи операторів у бд.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення статусу 200.

Отримання списку користувачів, яки є у складі групи:

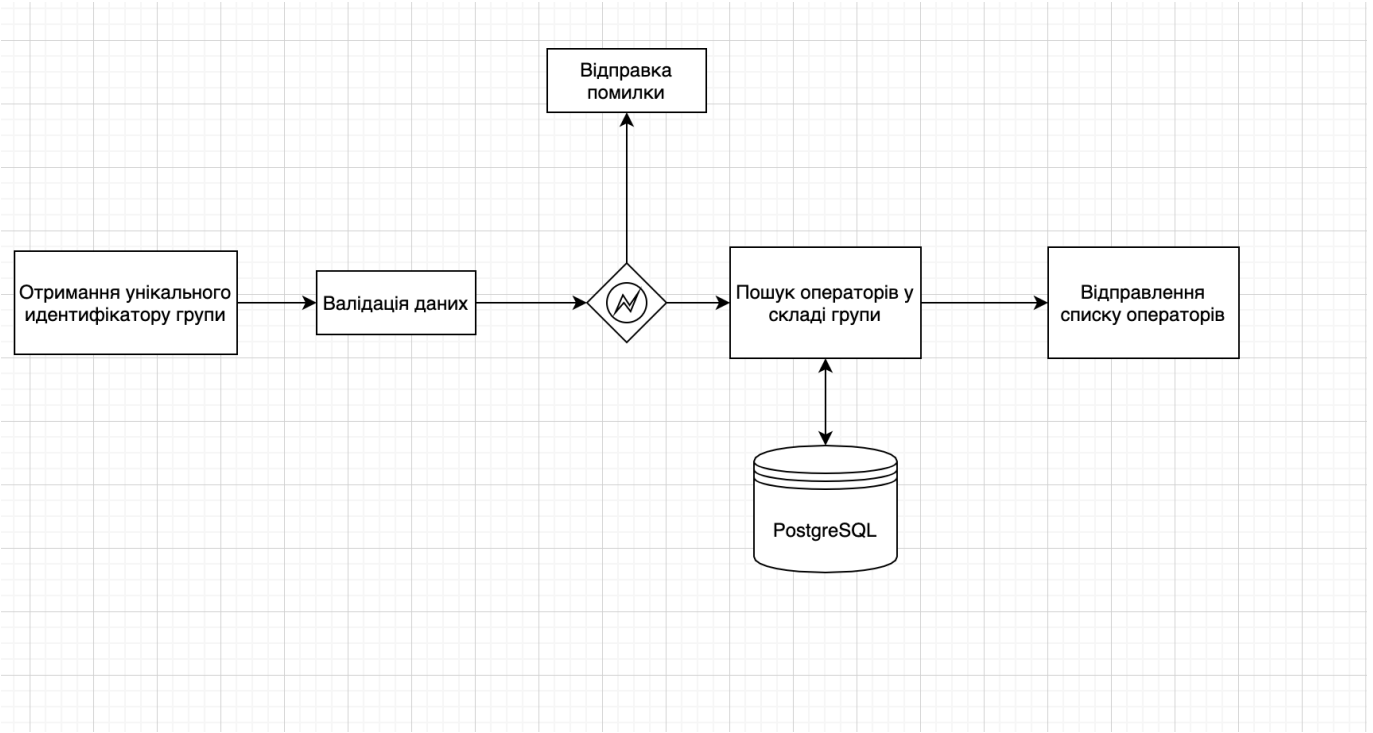


Рис. 3.20 - Отримання списку користувачів, яки є у складі групи

Алгоритм:

1. Отримання унікального ідентифікатору групи.
2. Обробка даних на сервері.
3. Пошук операторів, які є у складі групи.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення списку знайденних операторів.

Отримання списку продуктів, які є у складі групи:

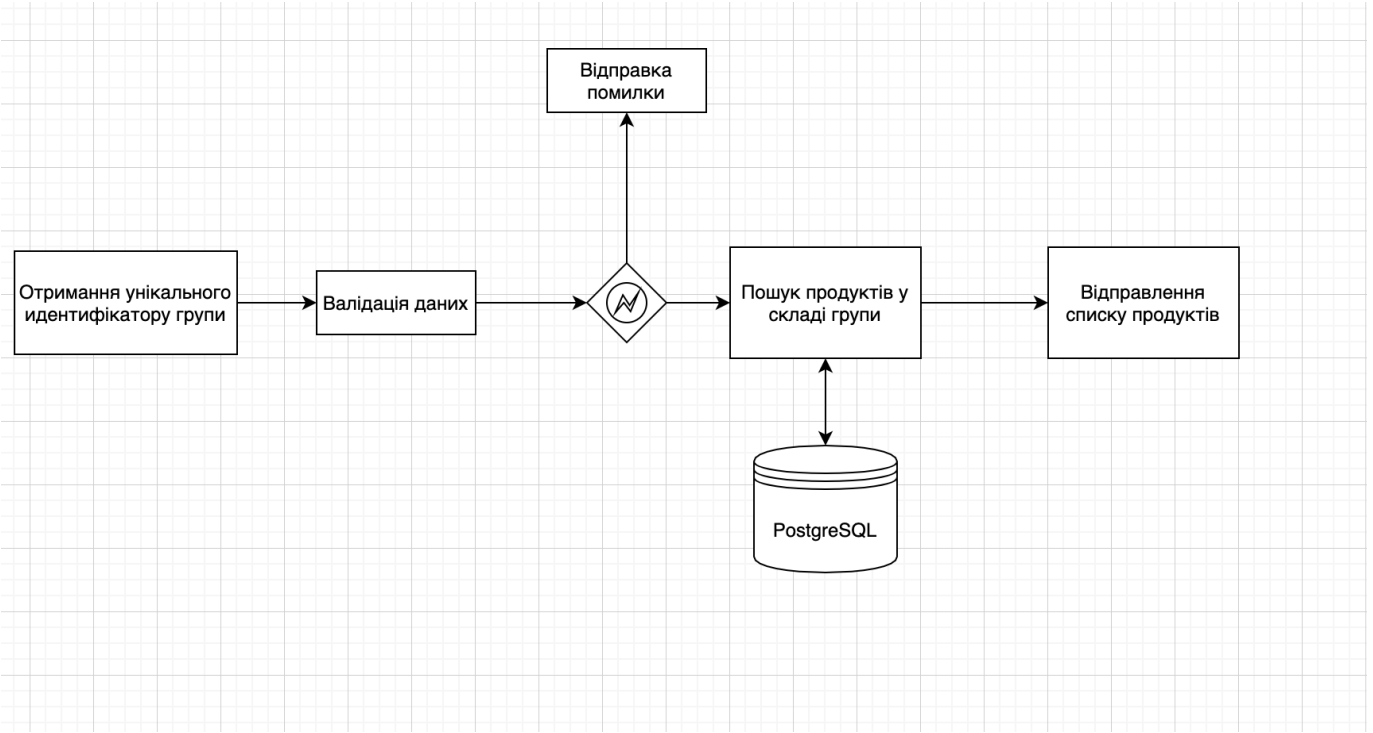


Рис. 3.21 - Отримання списку продуктів, які є у складі групи

Алгоритм:

1. Отримання унікального ідентифікатору групи.
2. Обробка даних на сервері.
3. Пошук продуктів, які є у складі групи.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення списку знайденних продуктів.

Додавання користувача у групу:

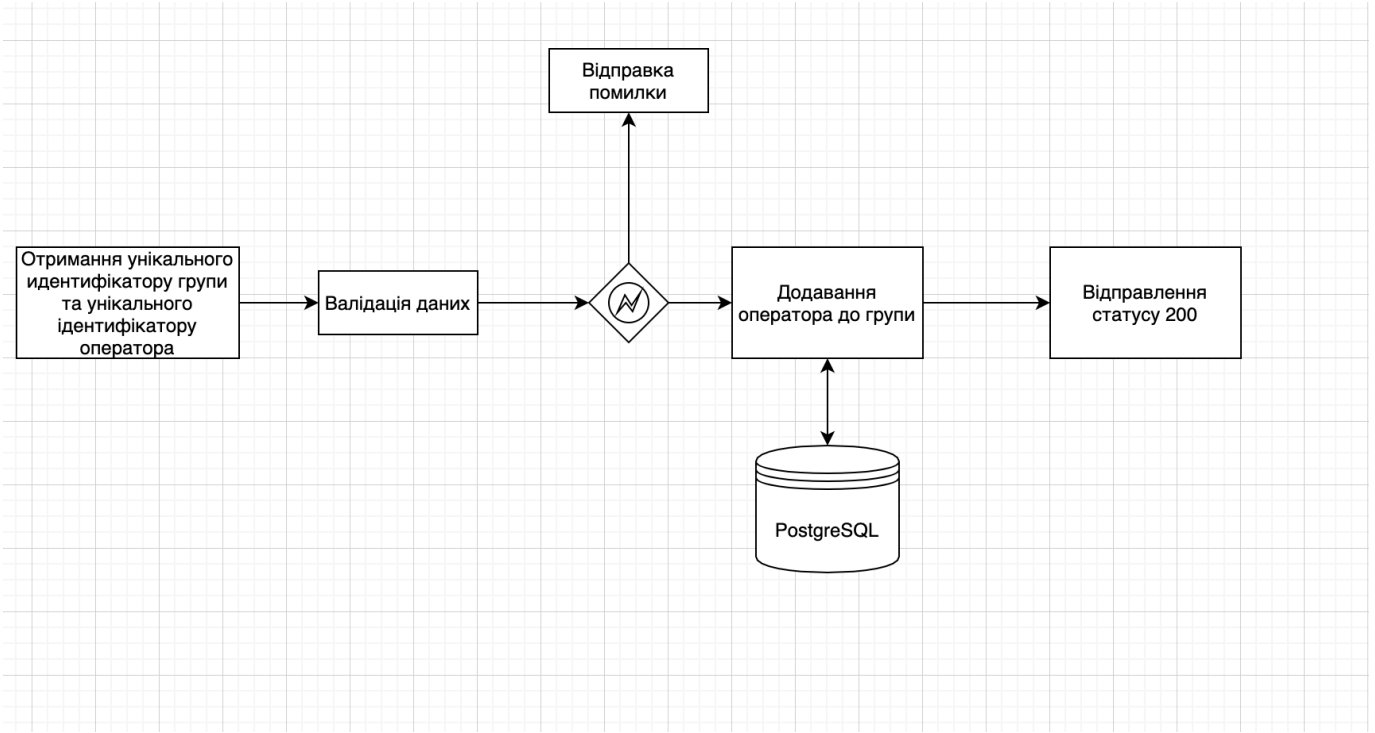


Рис. 3.22 - Додавання користувача у групу

Алгоритм:

1. Отримання унікального ідентифікатору групи та унікального ідентифікатору оператора.

2. Обробка даних на сервері.

3. Додавання оператора у групу.

4. Якщо помилка — то повернення помилки.

5. Інакше — відправлення статусу 200.

Видалення користувача з групи:

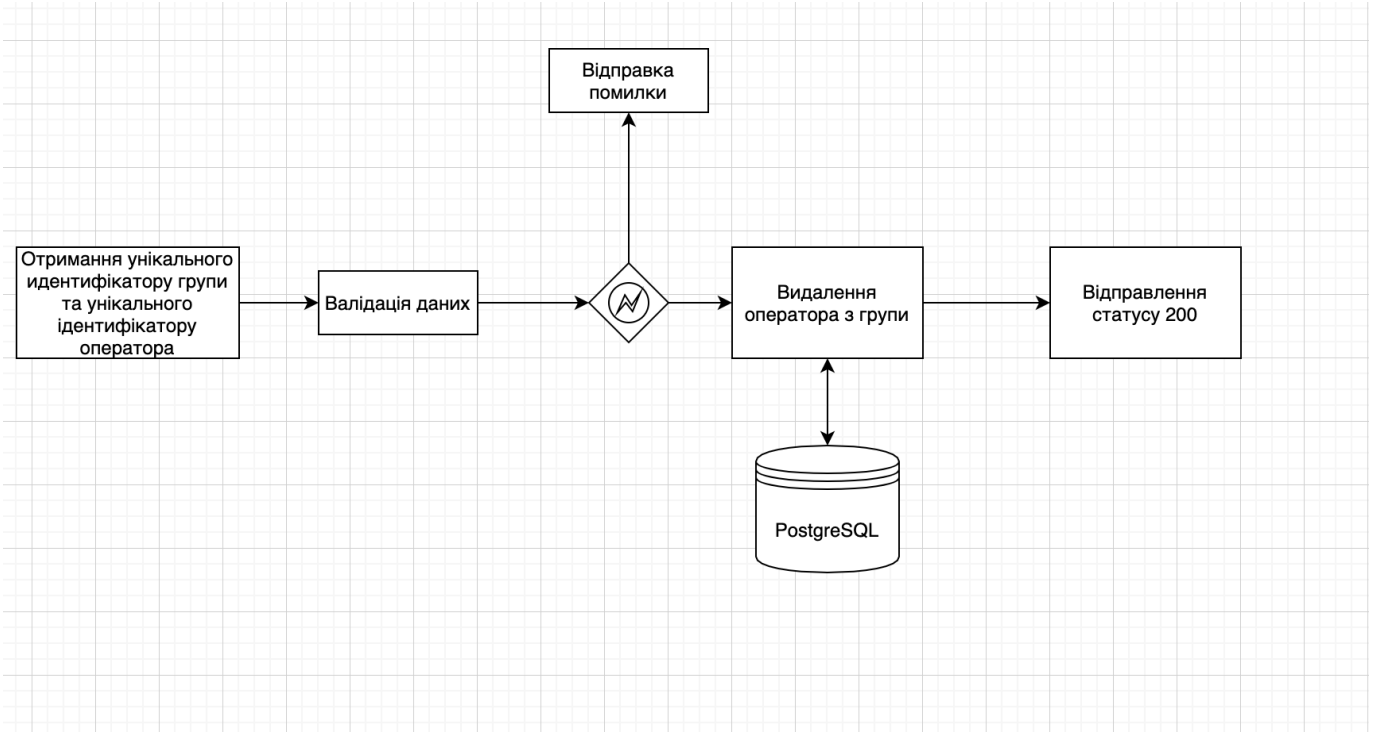


Рис. 3.23 - Видалення користувача з групи

Алгоритм:

1. Отримання унікального ідентифікатору групи та унікального ідентифікатору оператора.

2. Обробка даних на сервері.

3. Видалення оператора у групу.

4. Якщо помилка — то повернення помилки.

5. Інакше — відправлення статусу 200.

Додавання продукту у групу:

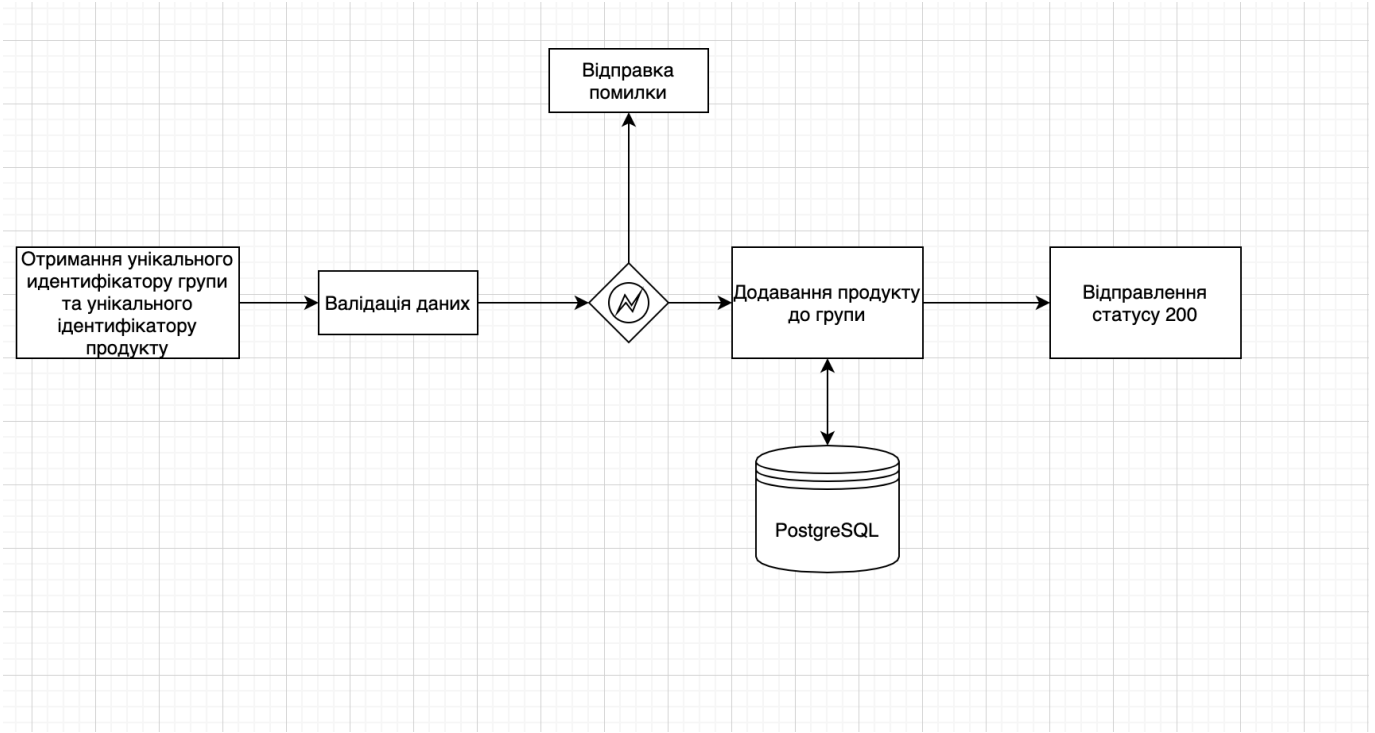


Рис. 3.24 - Додавання продукту у групу

Алгоритм:

1. Отримання унікального ідентифікатору групи та унікального ідентифікатору продукту.

2. Обробка даних на сервері.

3. Додавання продукту у групу.

4. Якщо помилка — то повернення помилки.

5. Інакше — відправлення статусу 200.

Видалення продукту з групи:

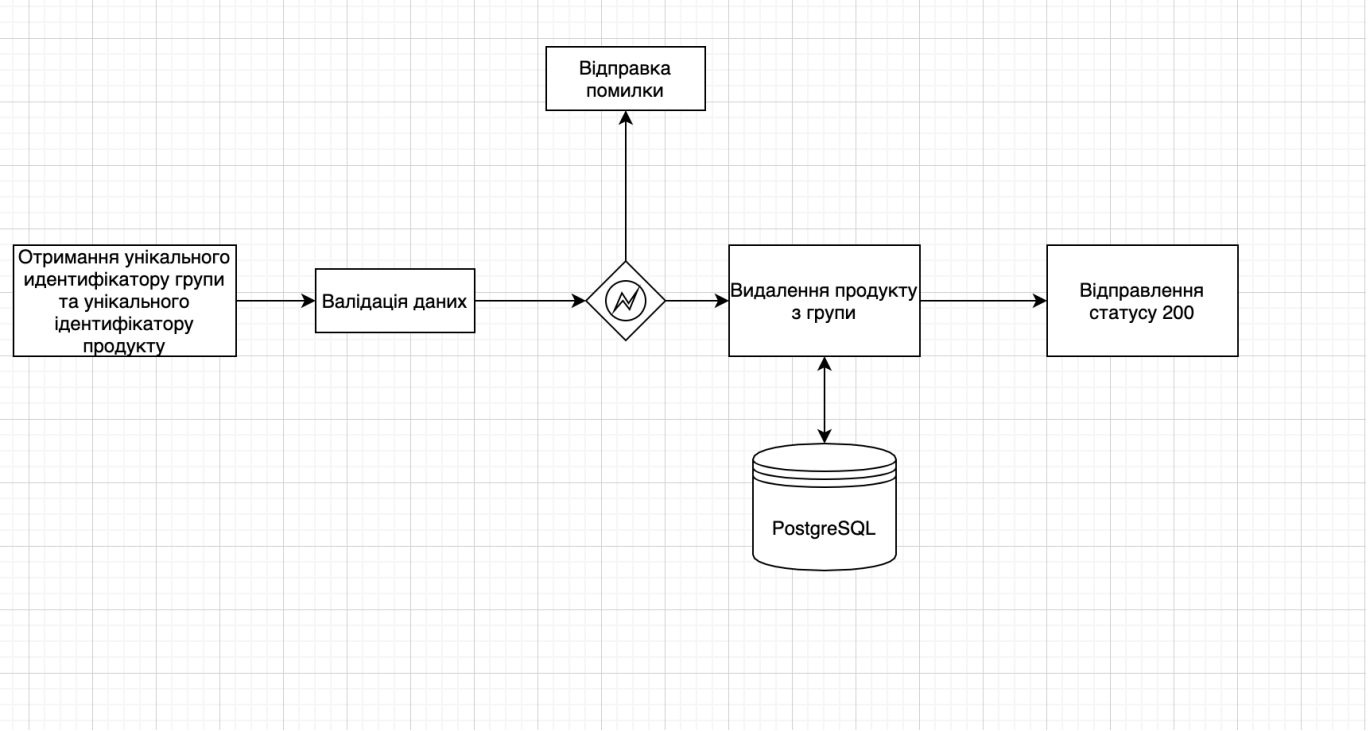


Рис. 3.25 - Видалення продукту з групи

Алгоритм:

1. Отримання унікального ідентифікатору групи та унікального ідентифікатору продукту.

2. Обробка даних на сервері.

3. Видалення продукту у групу.

4. Якщо помилка — то повернення помилки.

5. Інакше — відправлення статусу 200.

Отримання фінансової звітности:

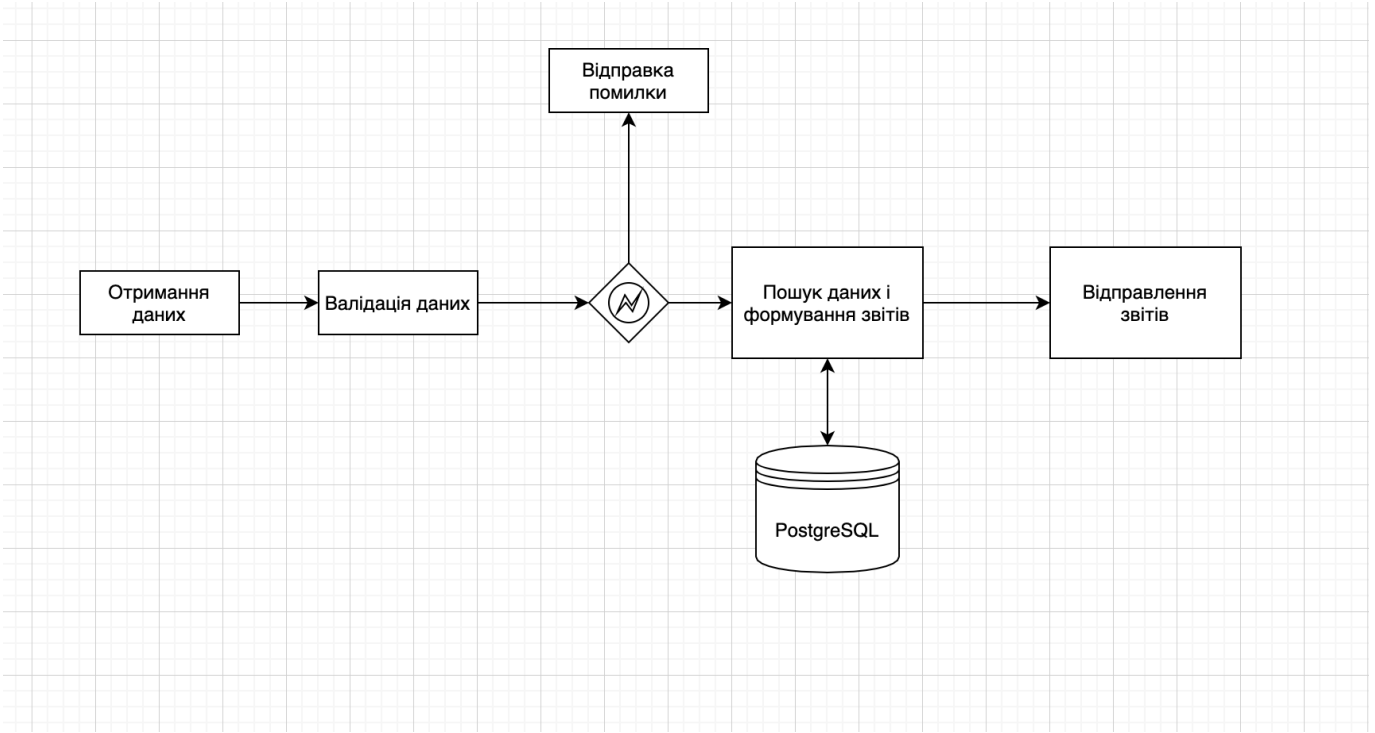


Рис. 3.26 - Отримання фінансової звітности

Алгоритм:

1. Отримання даних.
2. Обробка даних на сервері.
3. Пошук і формування звітів.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення звітів.

Експорт фінансової звітності у .csv:

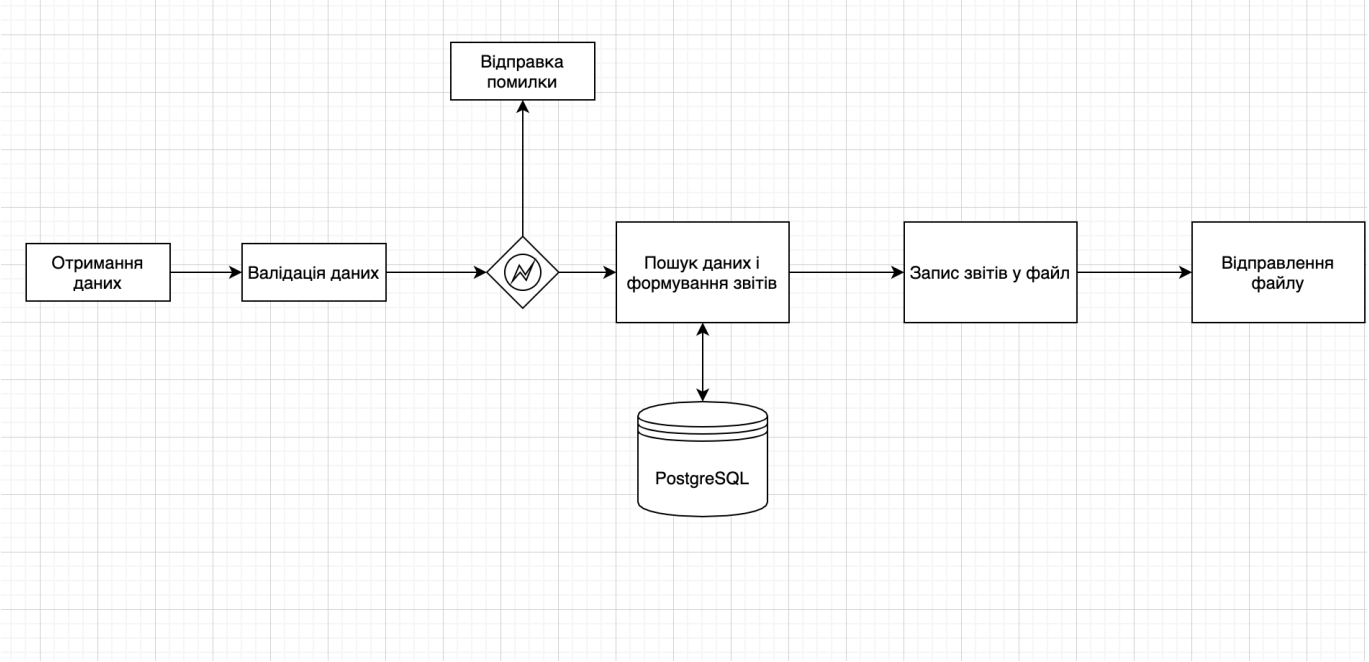


Рис. 3.27 - Експорт фінансової звітності у .csv

Алгоритм:

1. Отримання даних.
2. Обробка даних на сервері.
3. Пошук і формування звітів.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення файлу зі звітами.

Отримання детальної статистики по замовленням:

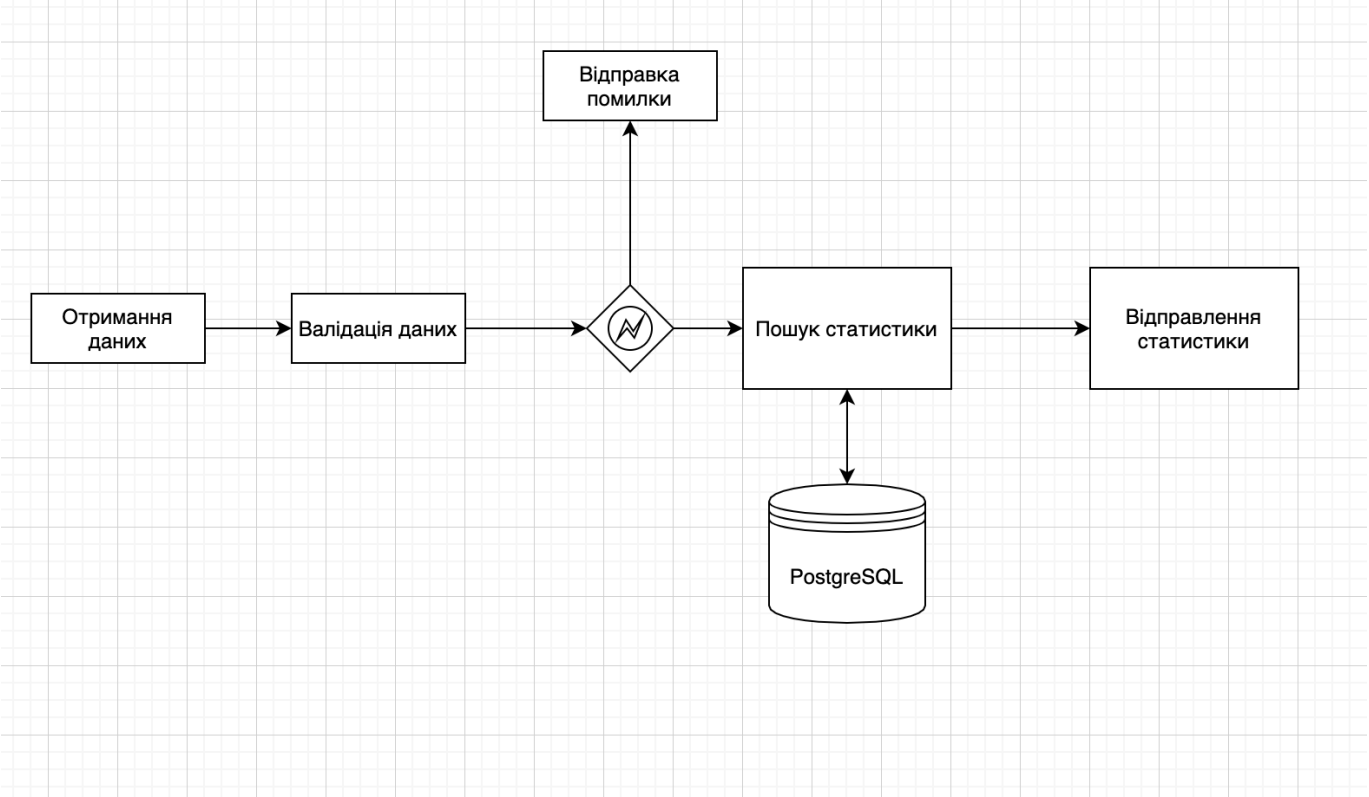


Рис. 3.28 - Отримання детальної статистики по замовленням

Алгоритм:

1. Отримання даних.
2. Обробка даних на сервері.
3. Пошук статистики.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення статистики.

Отримання замовлень через меню детальної статистики замовлень:

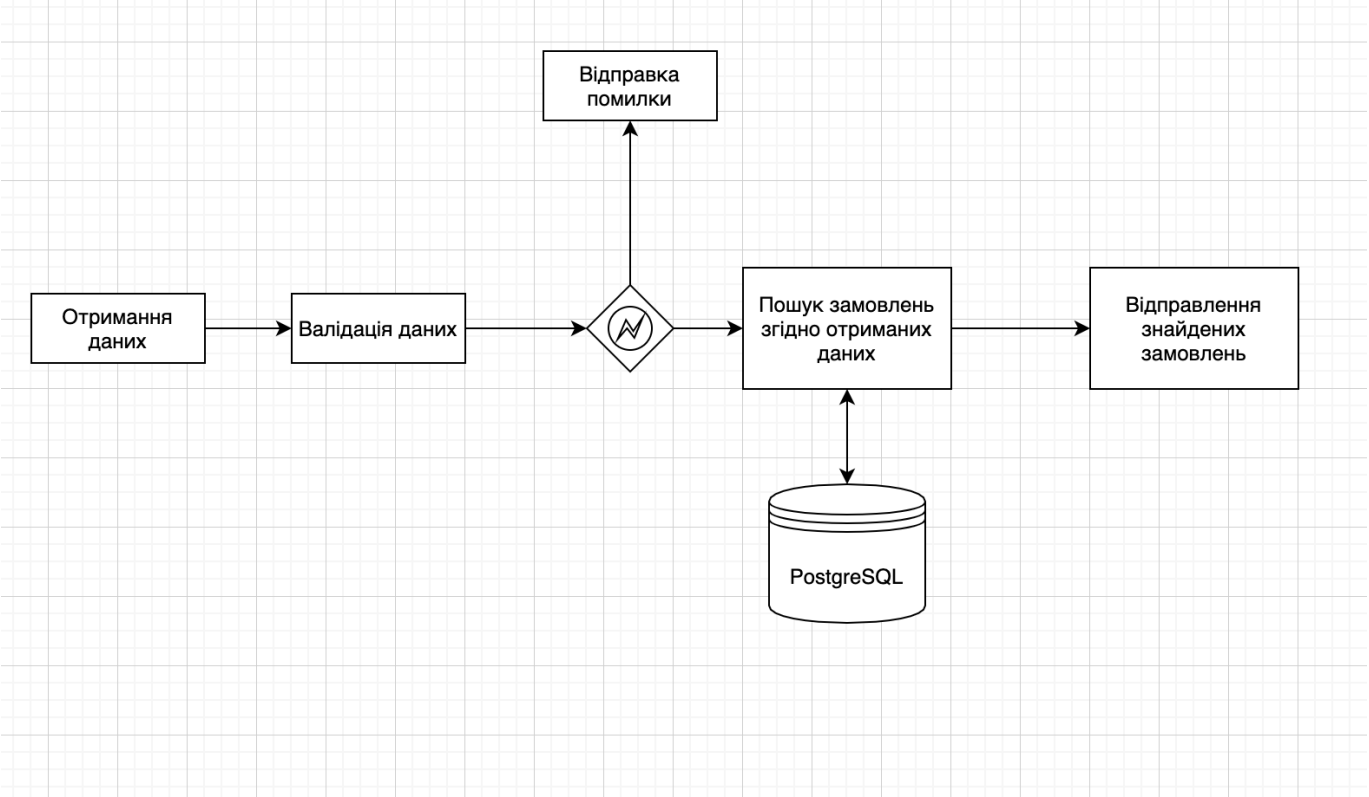


Рис. 3.29 - Отримання замовлень через меню детальної статистики замовлень

Алгоритм:

1. Отримання даних.
2. Обробка даних на сервері.
3. Пошук замовлень, згідно отриманих даних.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення знайдених замовлень.

Експорт замовлень у .csv:

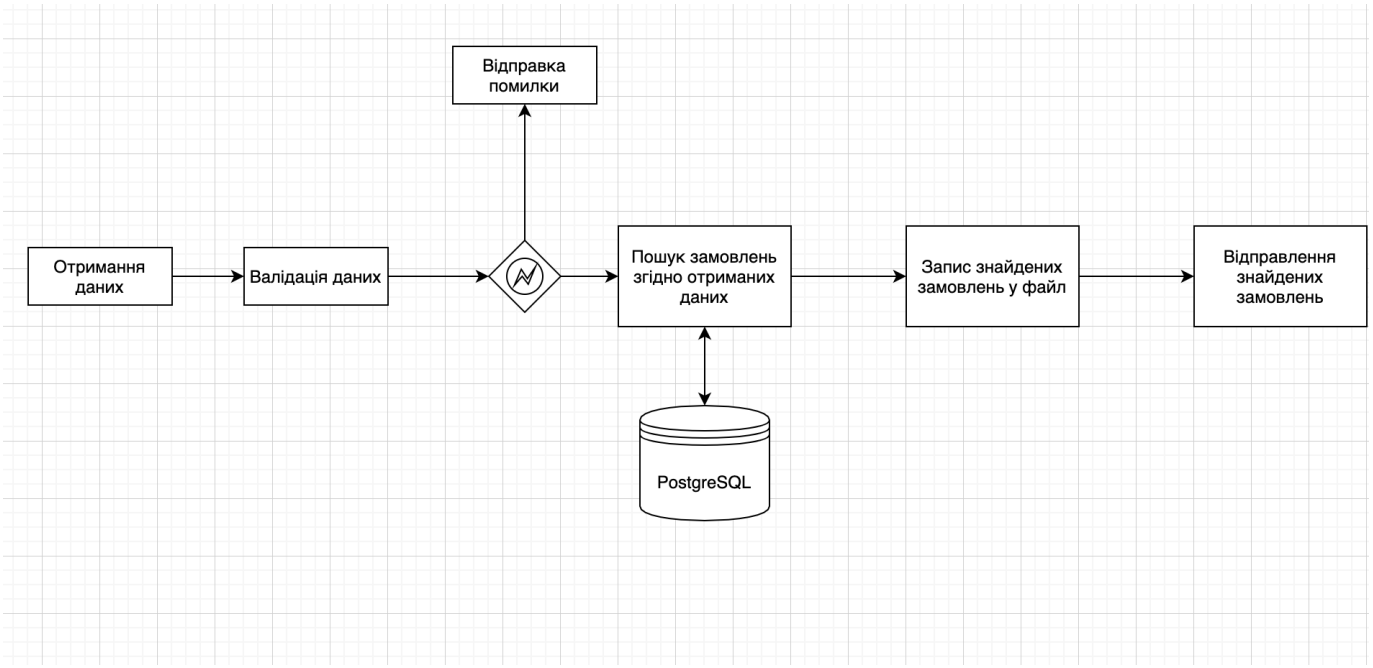


Рис. 3.30 - Експорт замовлень у .csv

Алгоритм:

1. Отримання даних.
2. Обробка даних на сервері.
3. Пошук замовлень, згідно отриманих даних.
4. Якщо помилка — то повернення помилки.
5. Інакше — відправлення файлу зі знайденими замовленями .

Для розробки проекту було обрано мову програмування Java. База даних працює на PostgreSQL.

Також проект використовує такі технології, як Spring Data JDBC, Spring Security, Web Socket.

Spring забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники і організації, які хочуть створити інформаційну систему, засновану на платформі Java. Через широку функціональності важко визначити найбільш значущі структурні елементи, з яких він складається. Spring не повністю пов'язаний з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

Ідея Spring Data JDBC полягає в тому, щоб надати доступ до реляційних баз даних без використання всієї складності JPA.

Spring Data JDBC фокусується на набагато більш простої моделі. Чи не буде кешування, відстеження змін, або ледачою завантаження. Замість цього, SQL запити будуть виконані тоді і тільки тоді, коли ви викликали метод сховища. Повертається результат буде повністю завантажений в пам'ять після виконання методу. Чи не буде і механізму "сесії" або проксі-об'єктів для entities. І все це повинно зробити Spring Data JDBC простішим і зрозумілим інструментів для доступу до даних.(habr.com)

Проект має лише один контроллер(ManagerController), який відповідає за сторінку адміністратора.

Опис сервісів, які використовує доданок, наведено в наступній таблиці:

Таб. 3.1 - Опис сервісів, які використовує доданок

Назва сервісу	Опис сервісу
SendingService	Відповідає за налаштування та розсилку повідомлень клієнтам.
LoggingService	Відповідає за налаштування логування та за саме логування дій користувача.
QualityService	Використовується для налаштування оцінювання та для самого оцінювання дзвінків.

Назва сервісу	Опис сервісу
StatisticService	Відповідає за збір, збегірання, обробку та демонстраці статистики по замовленням, користувачам і т.д.
WebSocketService	Відповідає за налаштування з’єднання по веб-сокетам.
ShipperService	Відповідає за редагування налаштувань доставки товару.
OrderService	Відповідає за релагування замовлень, налаштування роботи з замовленнями, створення замовлень.
CustomerService	Відповідає за редагування інформації про клиєнтів, роботу з клієнтами.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було змодельовано і спроектовано основні кейси програмного забезпечення та описано основні модулі розробляемого програмного забезпечення. Для розробки використано фреймворк Spring framework.

РОЗДІЛ 4

ОГЛЯД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Огляд програмного забезпечення

Аутентифікація користувача:

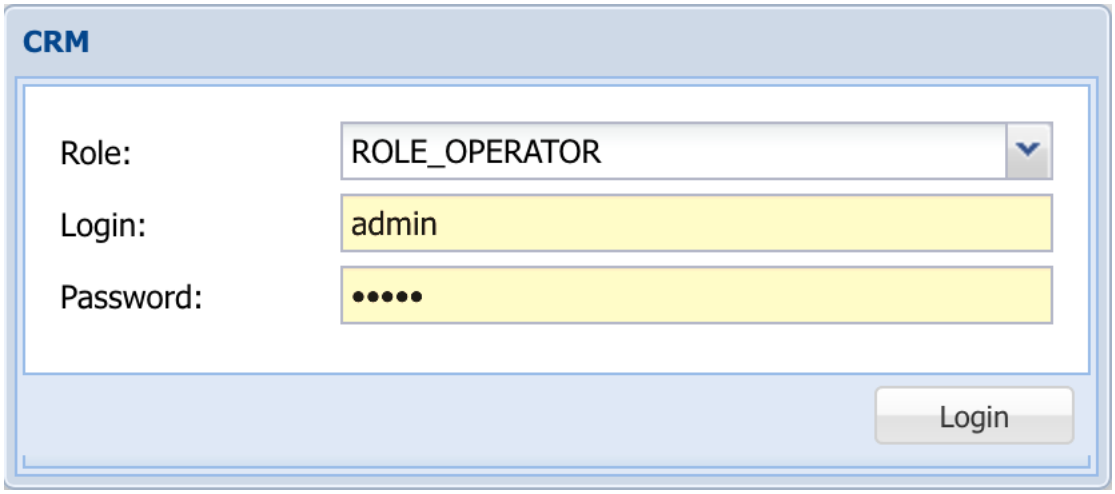


Рис. 4.1 - Аутентифікація користувача

Меню керування аудіозаписами дзвінків:

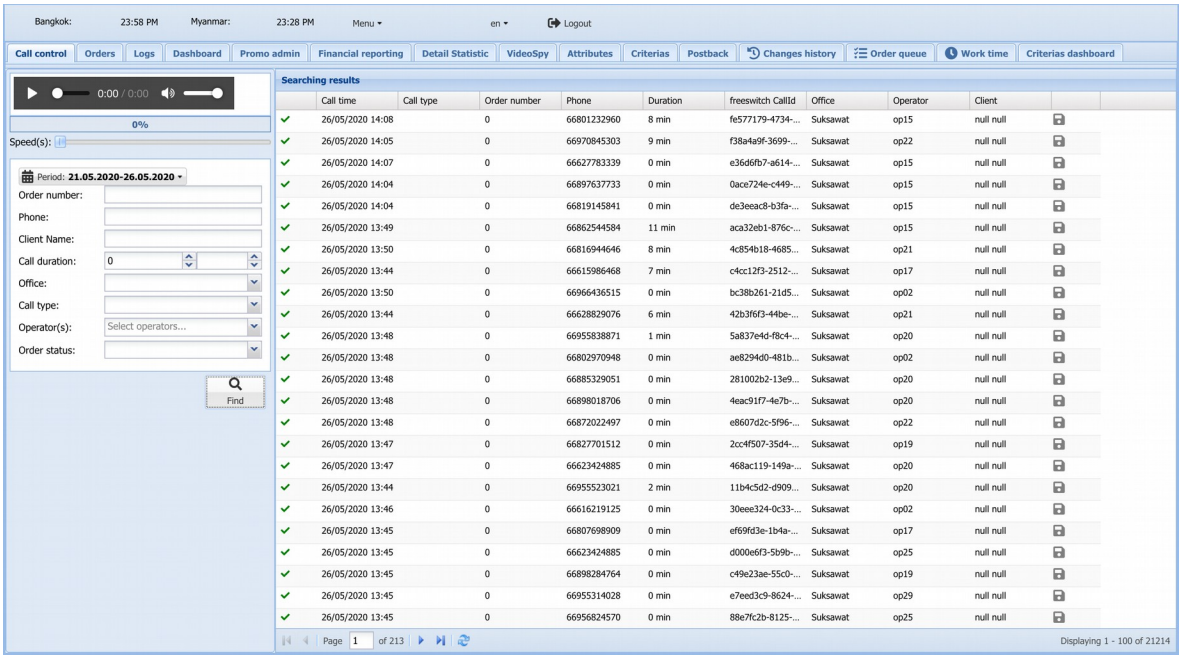


Рис. 4.2 - Меню керування аудіозаписами дзвінків

Меню керування замовленнями:

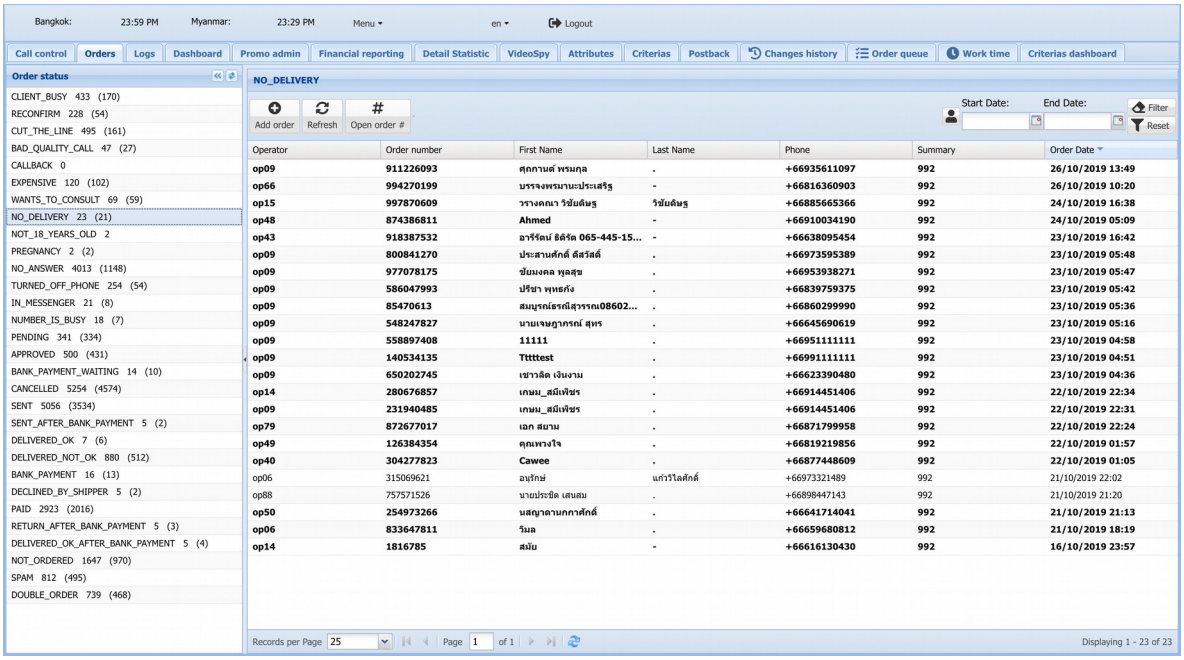


Рис 4.3 - Меню керування замовленнями

Меню керування шаблонами для логування дій користувача:

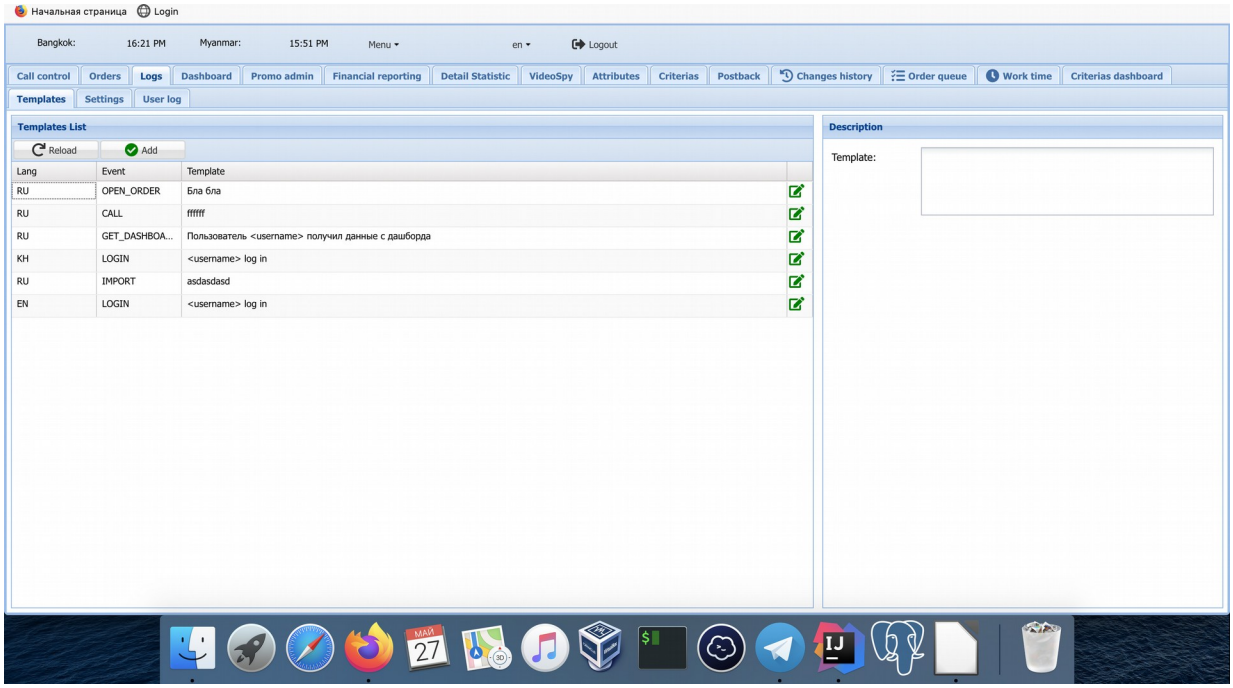


Рис. 4.4 - Меню керування шаблонами для логування дій користувача

Меню для налаштування подій логування:

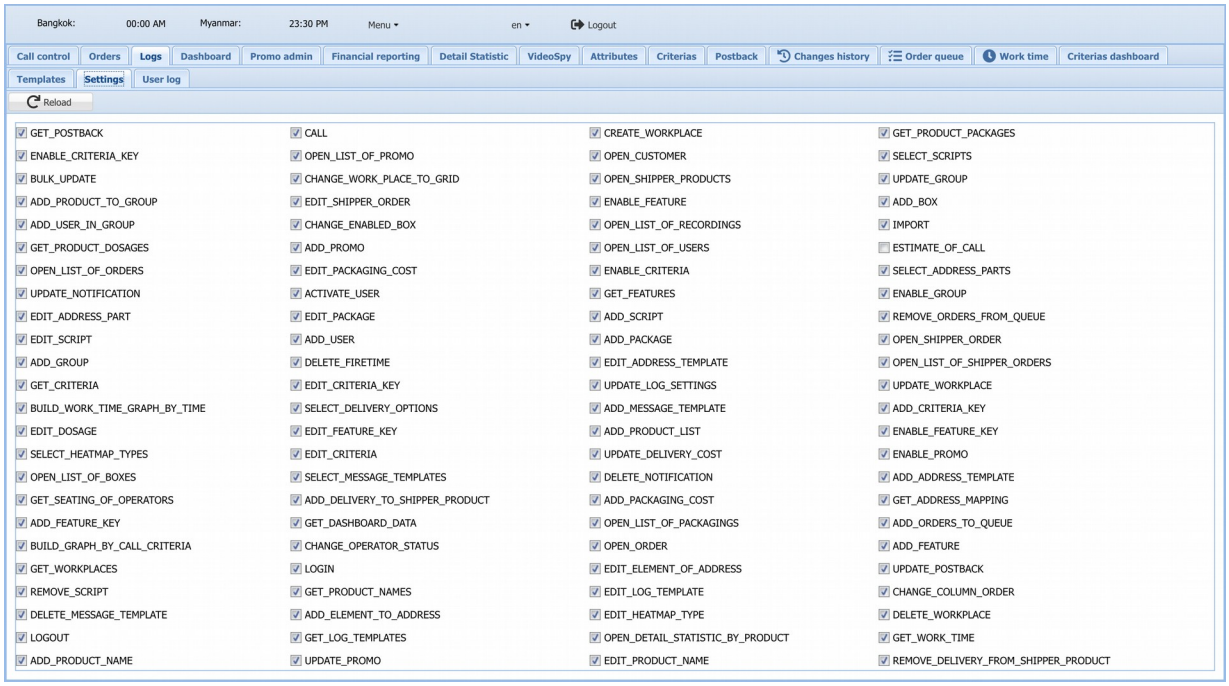


Рис. 4.5 - Меню для налаштування подій логування

Перегляд логів дій користувача:

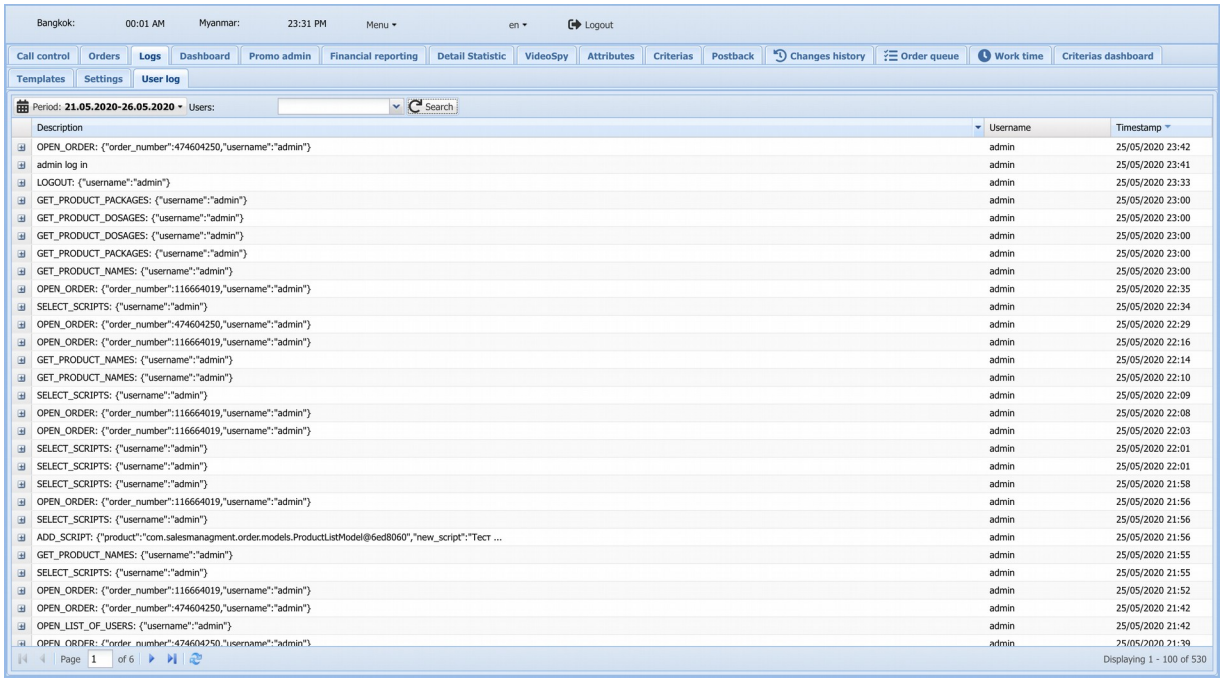


Рис. 4.6 - Перегляд логів дій користувача

Перегляд поточної статистики операторів:

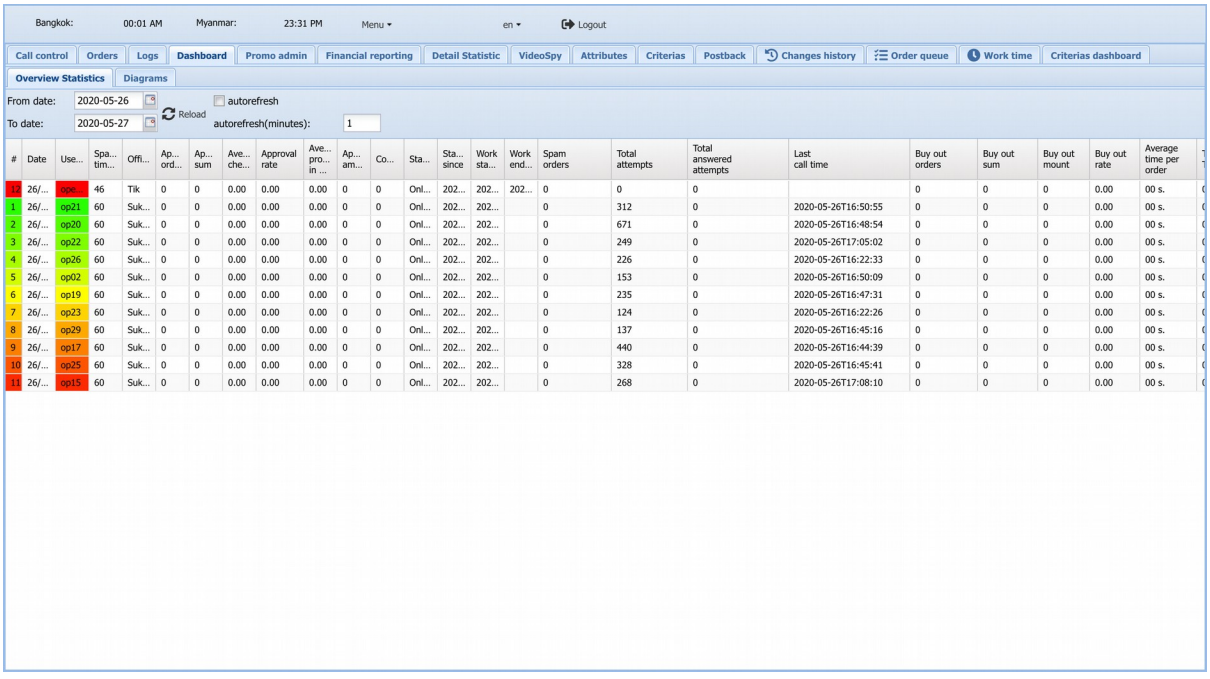


Рис. 4.7 - Перегляд поточної статистики операторів

Меню налаштування промо-акцій:

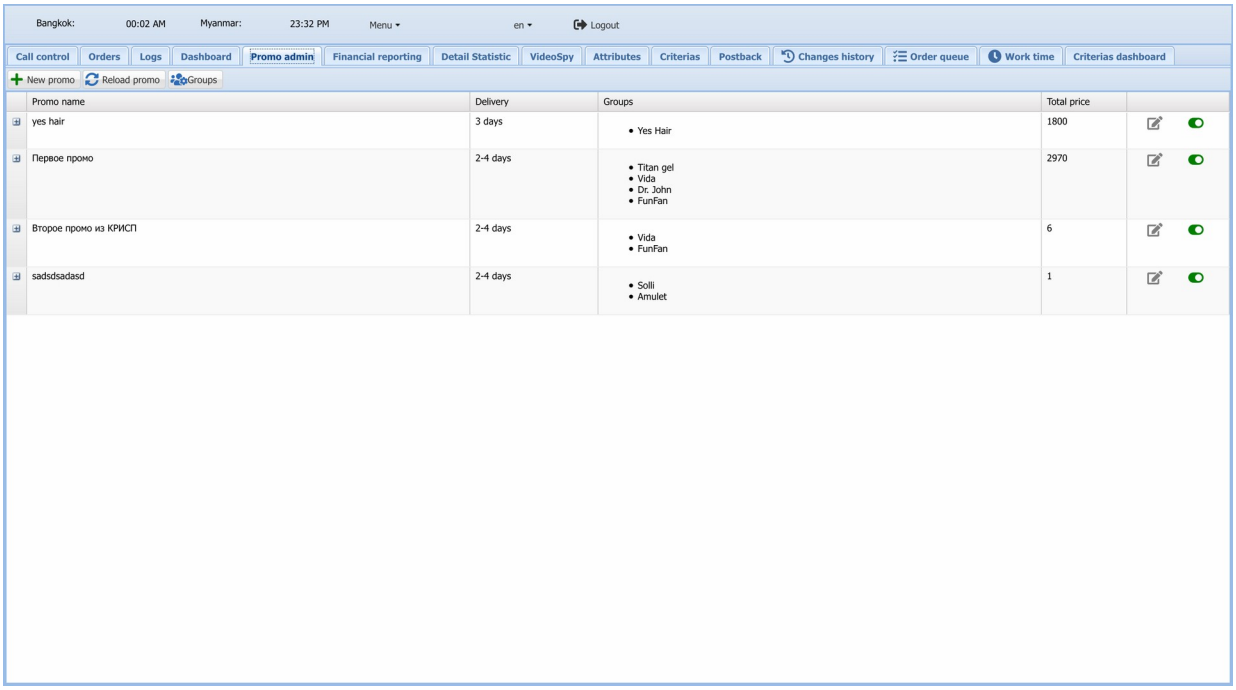


Рис. 4.8 - Меню налаштування промо-акцій

Перегляд фінансових звітів:

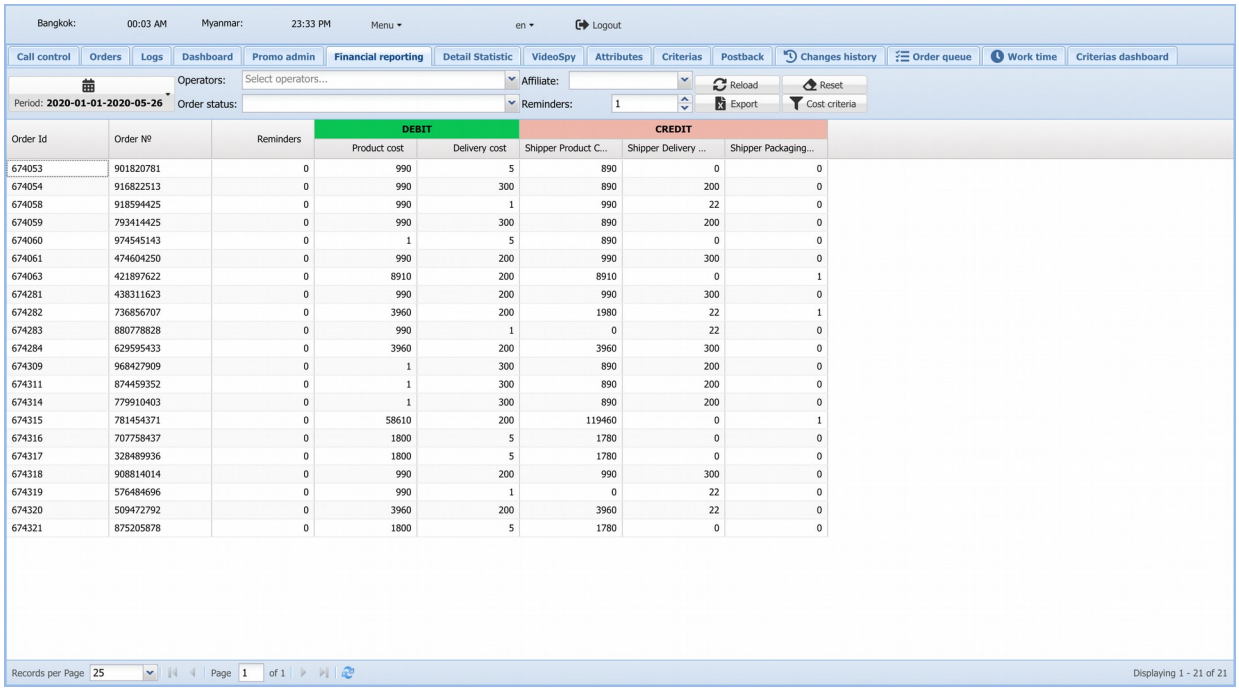


Рис 4.9 - Перегляд фінансових звітів

Меню перегляду детальної статистики по замовленням та перегляду замовлень по статистиці:

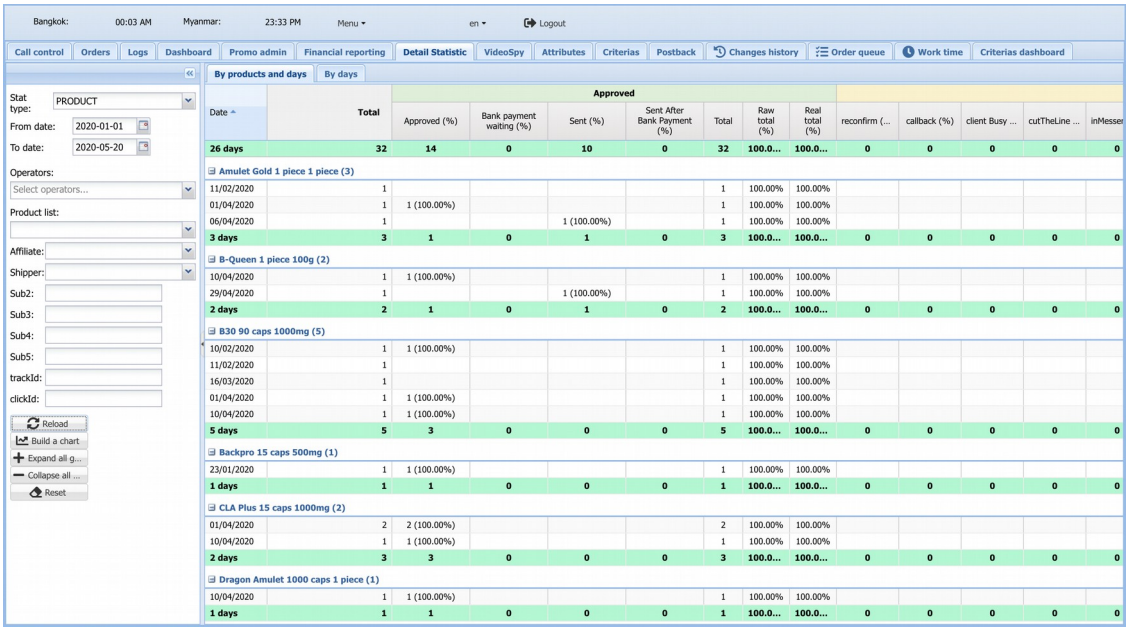


Рис. 4.10 - Меню перегляду детальної статистики по замовленням та перегляду замовлень по статистиці

Меню налаштування атрибутів для формування інформації про клієнта:

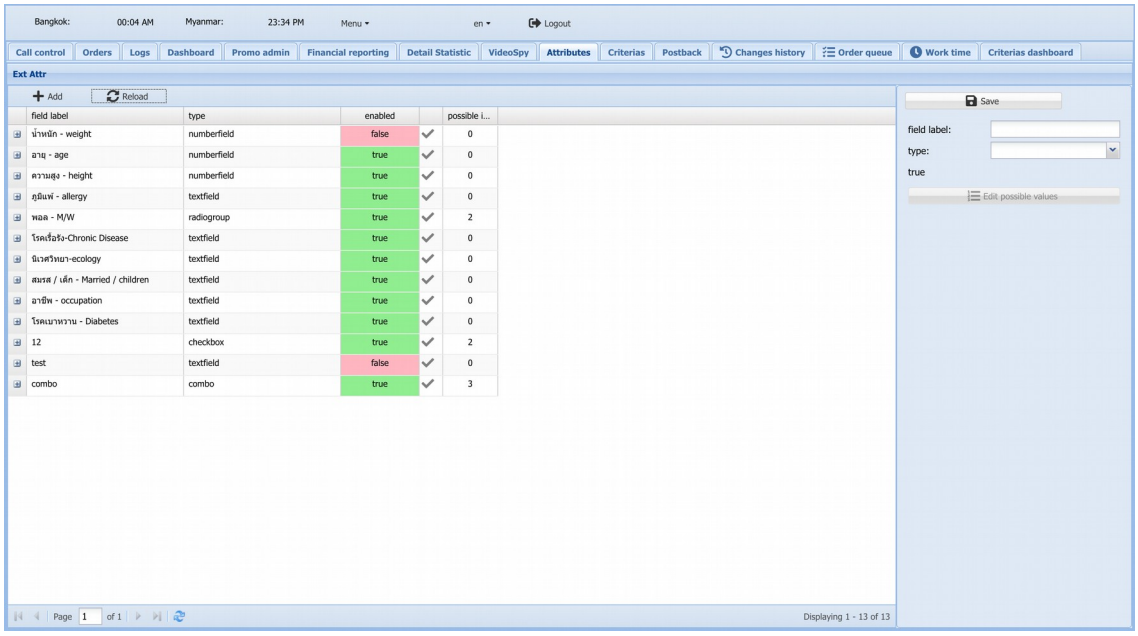


Рис. 4.11 - Меню налаштування атрибутів для формування інформації про клієнта

Меню налаштування критеріїв для оцінки дзвінків:

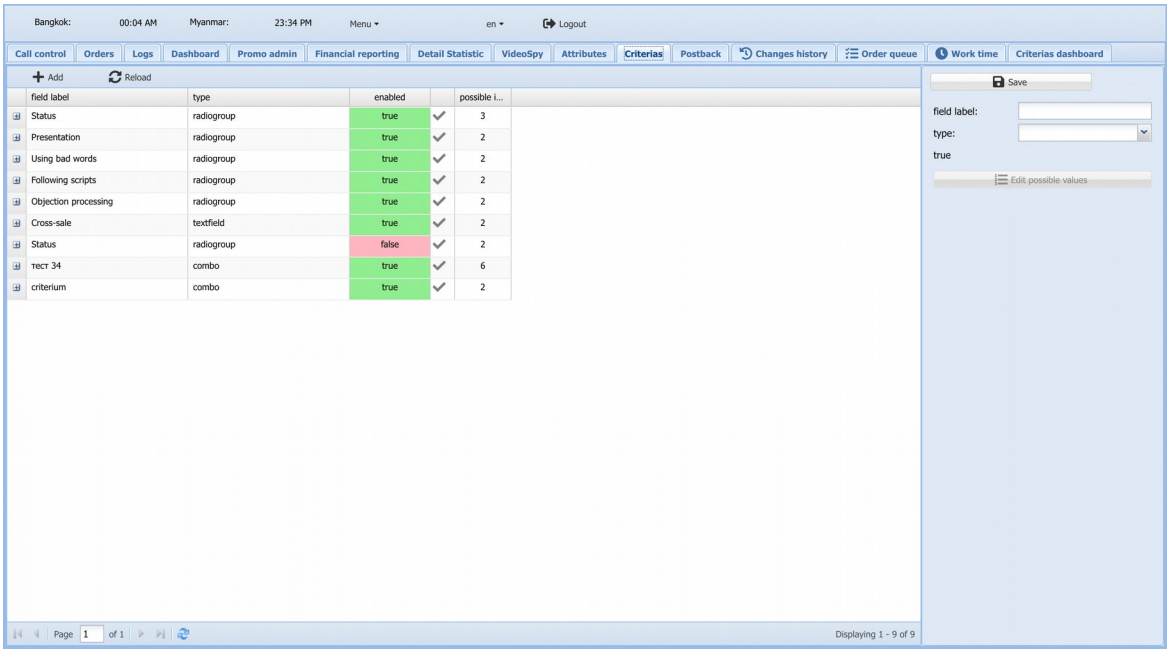


Рис. 4.12 - Меню налаштування критеріїв для оцінки дзвінків

Меню перегляду історії зміни замовлень:

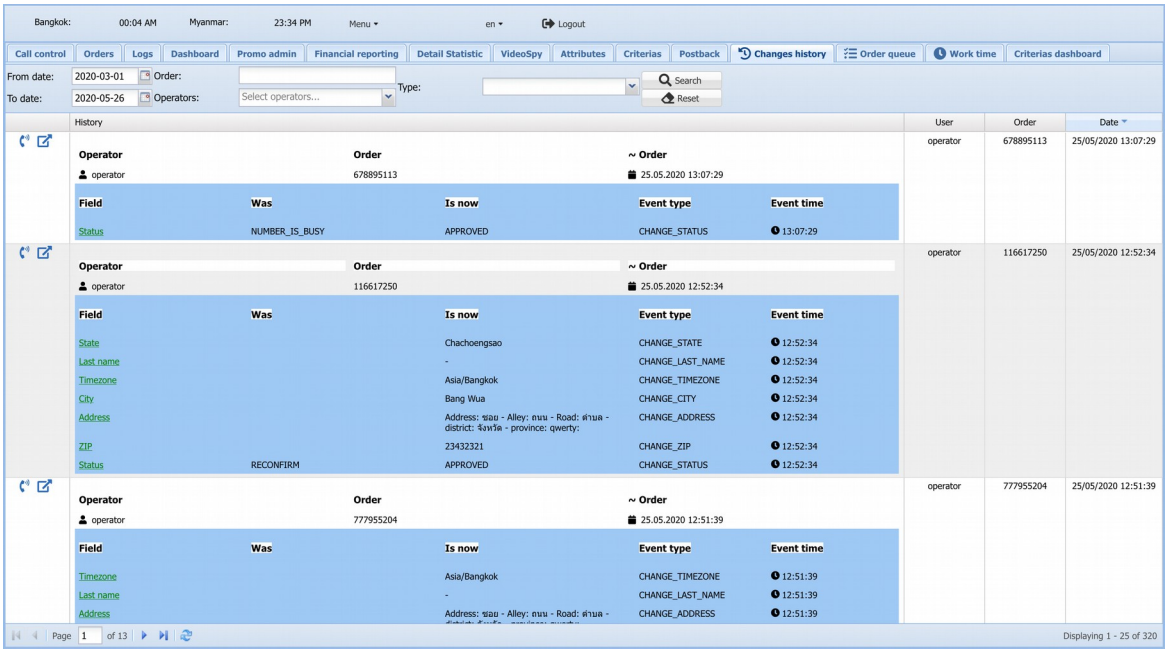


Рис. 4.13 - Меню перегляду історії зміни замовлень

Меню ручного керування чергою замовлень:

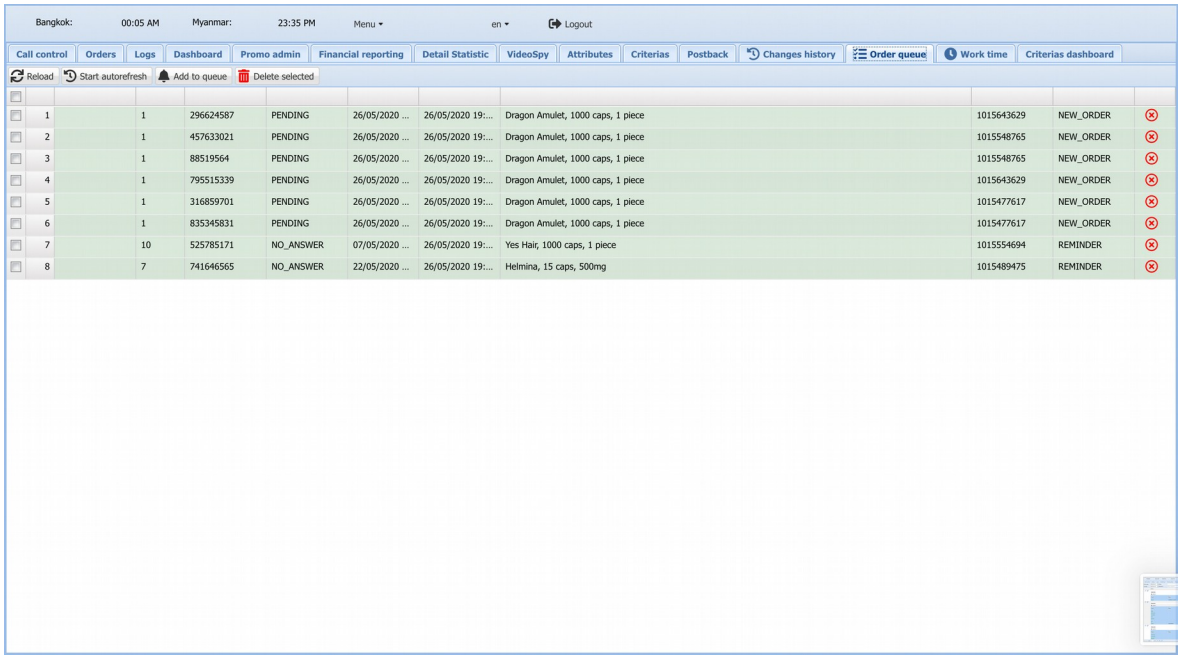


Рис. 4.14 - Меню ручного керування чергою замовлень

Меню перегляду робочого часу операторів:

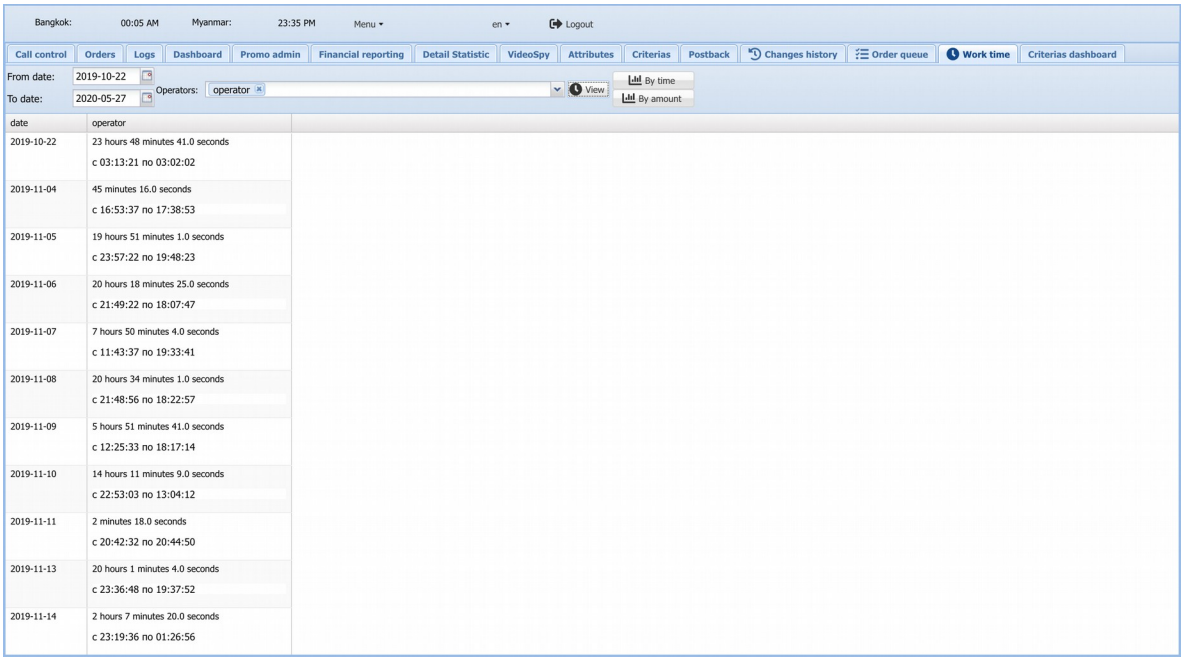


Рис. 4.15 - Меню перегляду робочого часу операторів

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі було оглянуто та продемонстровано роботу програмного додатку, створеного у ході розробки дипломного проекту. Усі умови, які були раніше сформульовані та спроектовані — виконано.

ВИСНОВОК

У ході роботи над дипломним проектом було спроектовано та розроблено модуль для налаштування системі обробки запитів колл-центу. Було сформульовано, проаналізовано та виконано усі поставлені задачі. Також було створено усю необхідну документацію, проведено огляд програмного доданку та проаналізовано аналогічні існуючі рішення

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Система управления взаимоотношениями с клиентами. URL: https://ru.wikipedia.org/wiki/Система_управления_взаимоотношениями_с_клиентами

2. История развития CRM. URL: http://www.voit.pro/blog/istoriya-razvitiya-crm/

3. Как реализованы CRM-процессы в типовых конфигурациях. URL: https://tqm.com.ua/likbez/crm/crm-protsessy-v-konfiguratsiyah-1c-8#klienty.

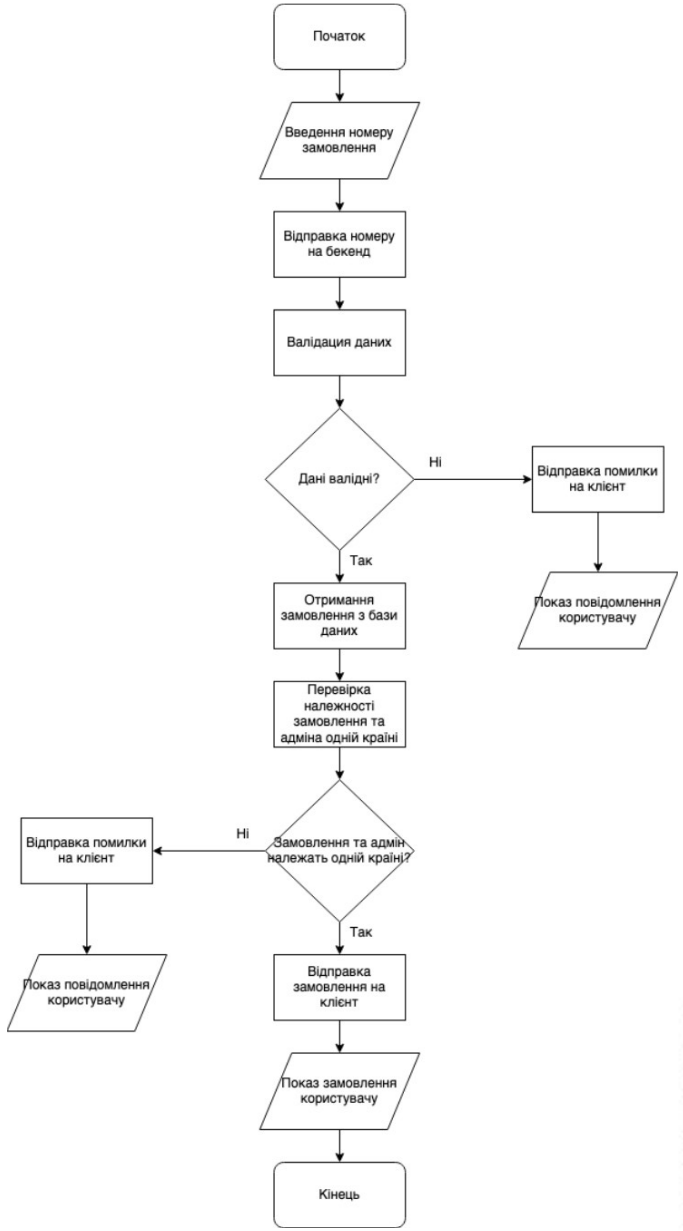
4. Проектирование программного обеспечения URL: https://ru.wikipedia.org/wiki/Проектирование_программного_обеспечения.

ГРАФІЧНІ МАТЕРІАЛИ

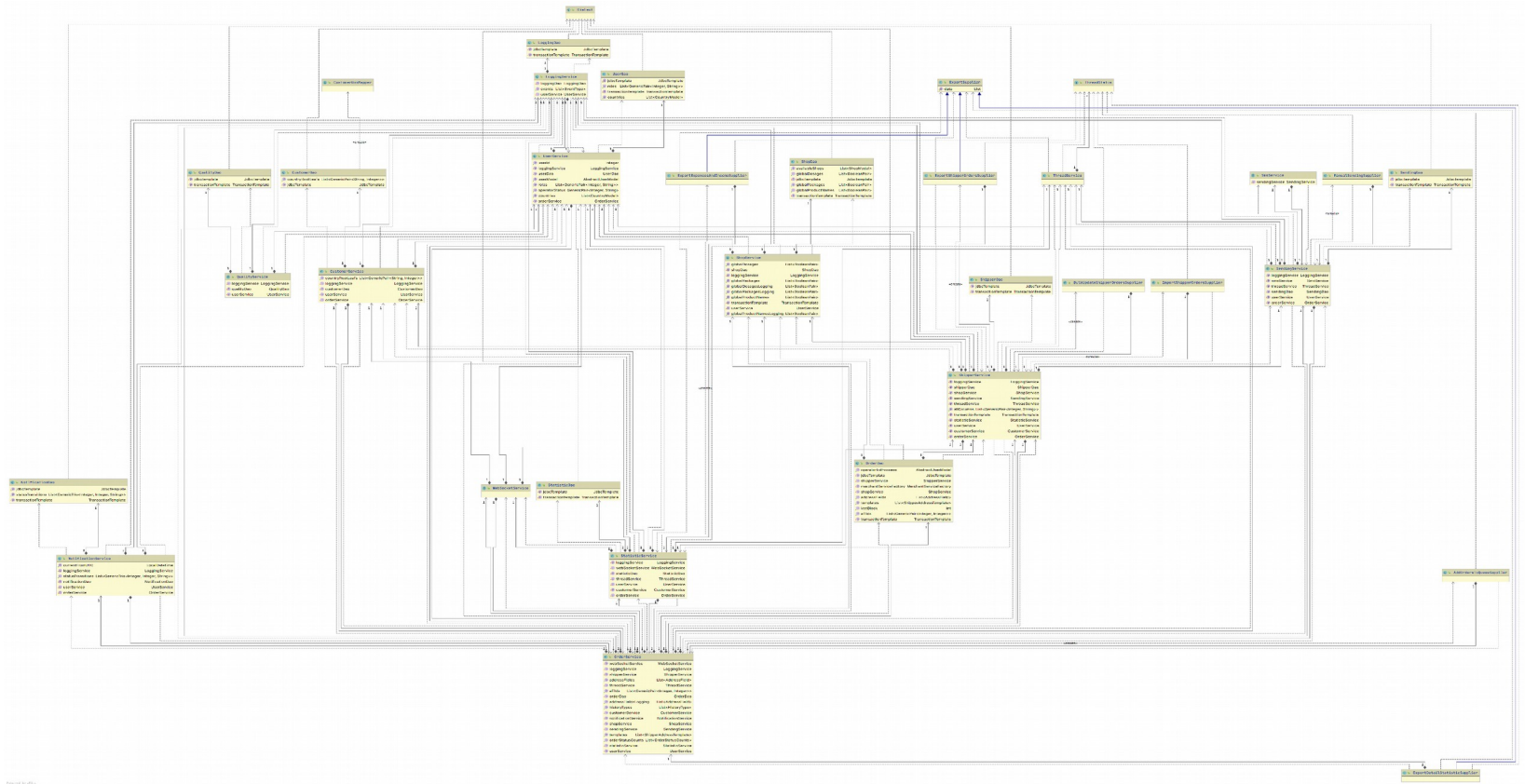
ДО ДИПЛОМНОГО ПРОЕКТУ

на тему: “Розробка модулю налаштування програми обробки заявок колл-центу”

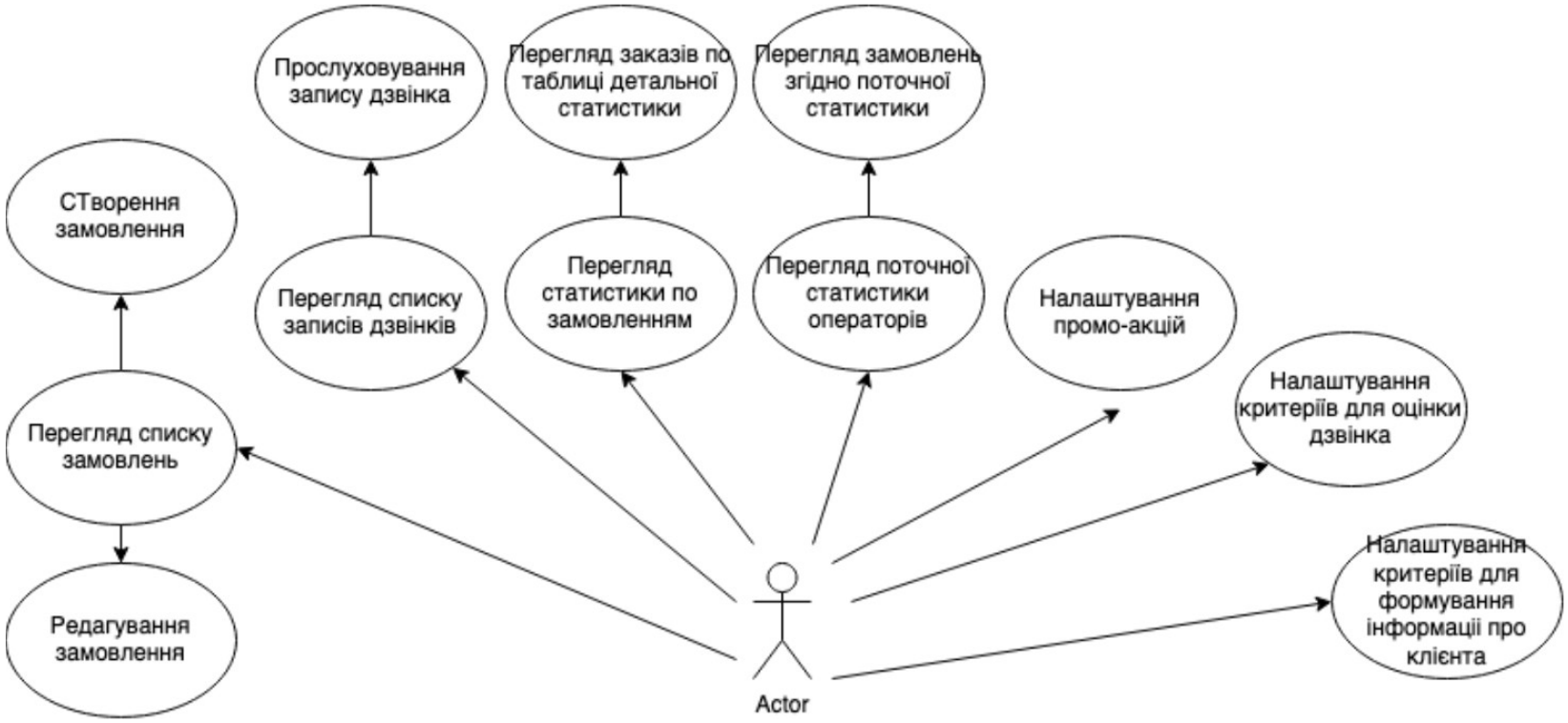
Київ - 2020



Зм.	Арк.	№ докум.	Підпис	Дата	Розробка модулю налаштування программи обробки заказів колл центру	Літ.	Маса	Масштаб
Розроб.		Маркушевський Р. С.						
Перевір.		Сімоненко А.В.				Арк. 1	Аркушів 1	
Н. контр.		Сімоненко В.П.			Дипломний проект	КПІ ФІОТ кафедра ОТ гр. ІП-64		
Затверд.		Стіренко С.Г.						



Зм.	Арк.	№ докум.	Підпис	Дата	Розробка модулю налаштування программи обробки заказів колл центру	Літ.	Маса	Масштаб
Розроб.		Маркушевський Р. С.						
Перевір.		Сімоненко А.В.				Арк. 1	Аркушів 1	
Н. контр.		Сімоненко В.П.			Дипломний проект	КПІ ФІОТ кафедра ОТ гр. ІП-64		
Затверд.		Стіренко С.Г.						



Зм.	Арк.	№ докум.	Підпис	Дата	Розробка модулю налаштування программи обробки заказів колл центру	Літ.	Маса	Масштаб
Розроб.		Маркушевський Р. С.						
Перевір.		Сімоненко А.В.				Арк. 1	Аркушів 1	
Н. контр.		Сімоненко В.П.			Дипломний проект	КПІ ФІОТ кафедра ОТ гр. ІП-64		
Затверд.		Стіренко С.Г.						

ДОДАТОК А

Система формування звітів

Текст програми

ІАЛЦ.467200.007А1

Листів 47

Київ - 2020

ManagerController

```
package com.salesmanagment.web.controllers;

import com.salesmanagment.logging.models.EventType;
import com.salesmanagment.logging.services.LoggingService;
import com.salesmanagment.order.models.*;
import com.salesmanagment.order.models.history.OrderChangeAnalyzer;
import com.salesmanagment.order.models.tableview.TrackingTableViewModel;
import com.salesmanagment.order.services.CustomerService;
import com.salesmanagment.order.services.OrderService;
import com.salesmanagment.quality.service.QualityService;
import com.salesmanagment.sending.models.Message;
import com.salesmanagment.sending.models.MessageFromReport;
import com.salesmanagment.sending.models.MessageTemplate;
import com.salesmanagment.sending.models.Report;
import com.salesmanagment.sending.service.SendingService;
import com.salesmanagment.shipper.models.DeliveryModel;
import com.salesmanagment.shipper.models.Shipper;
import com.salesmanagment.shipper.models.client.ShipperProductSimpleModel;
import com.salesmanagment.shipper.services.ShipperService;
import com.salesmanagment.shops.models.GenericPair;
import com.salesmanagment.shops.models.GenericTrio;
import com.salesmanagment.shops.services.ShopService;
import com.salesmanagment.statistic.models.ProductStats;
import com.salesmanagment.statistic.models.SelectionTypeInfo;
import com.salesmanagment.statistic.services.StatisticService;
import com.salesmanagment.threads.suppliers.ManualSendingSupplier;
import com.salesmanagment.users.models.*;
import com.salesmanagment.users.service.UserService;
import com.salesmanagment.threads.ThreadService;
import com.salesmanagment.web.util.DbUtil;
import com.salesmanagment.web.util.JsonUtil;
import com.salesmanagment.websocket.services.WebSocketService;
import org.apache.log4j.Logger;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.stereotype.Controller;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.*;


@Controller
public class ManagerController {
    @RequestMapping(value = "/manager")
    public String manager() {
        return "com.salesmanagment.layout.manager";
    }
    @Autowired
    ShopService shopService;
```

```
    @Autowired
    UserService userService;
    @Autowired
    CustomerService customerService;
    @Autowired
    OrderService orderService;
    @Autowired
    ShipperService shipperService;
    @Autowired
    private WebSocketService webSocketService;
    @Autowired
    private StatisticService statisticService;
    @Autowired
    private QualityService qualityService;
    @Autowired
    private LoggingService loggingService;
    @Autowired
    private ThreadService threadService;
    @Autowired
    private SendingService sendingService;
    private Logger logger = Logger.getLogger(ManagerController.class);
    @ResponseBody
    @RequestMapping(value="/api/manager/deliverymethods/all", method =
RequestMethod.GET)
    public Map<String, ? extends Object> getDeliveryMethods(@RequestParam(value =
"onlyEnabled", required = false) boolean onlyEnabled) {
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
shipperService.getAvailableDeliveryModels(userService.getUserModel().getCountryId(),
onlyEnabled));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/deliverymethods/{product_list_id}/
{shipper_id}/", method = RequestMethod.GET)
    public Map<String, ? extends Object>
getDeliveryMethodsForProductlist(@PathVariable("product_list_id") int productListId,

@PathVariable("shipper_id") int shipperId) {
        logger.debug(String.format("productListId: %s, shipeprId: %s.",
productListId, shipperId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
shipperService.getAvailableDeliveryModels(productListId, shipperId));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
```

```
    @ResponseBody
    @RequestMapping(value="/api/manager/users/{id}/activate", method =
RequestMethod.POST)
    public Map<String, ?> enableUser(@PathVariable Integer id) {
        logger.debug(String.format("id: %s", id));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.activateUser(id, true);
            modelMap.put("success", true);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/users/{id}/deactivate", method =
RequestMethod.POST)
    public Map<String, ?> disableUser(@PathVariable Integer id) {
        logger.debug(String.format("id: %s", id));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.activateUser(id, false);
            modelMap.put("success", true);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/mappings", method =
RequestMethod.GET)
    public Map<String,?> getAddressMappings() {
        Map<String, Object> modelMap = new HashMap<>(3);
        Integer countryId = userService.getUserModel().getCountryId();
        try {
            List<AddressMapping> addressFields =
orderService.getAddressMappings(countryId);
            logger.debug(String.format("addressFields: %s", addressFields));
            if (addressFields != null) {
                logger.debug("addressFields != null");
                modelMap.put("success", true);
                modelMap.put("data", addressFields);
            }
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/mappings", method =
RequestMethod.POST)
    public Map<String,?> addAddressMapping(@RequestParam("address_part_id") Integer
addressPartId,
```

```
                                            @RequestParam("list_number") Integer
listNumber,
                                            @RequestParam("disabled") boolean
disabled) {
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("countryId: %s, adressPartId: %s, listNumber: %s,
disabled: %s", countryId, addressPartId, listNumber, disabled));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            orderService.addAddressMapping(countryId, addressPartId, listNumber,
disabled);
            modelMap.put("success", true);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/mappings/{id}", method =
RequestMethod.PUT)
    public Map<String,?> updateAddressMapping(@PathVariable("id") Integer id,
                                              @RequestBody MultiValueMap<String,
String> params) {
        logger.debug(String.format("id: %s", id));
        Map<String, Object> modelMap = new HashMap<>(3);
        Integer countryId = null;
        Integer addressPartId = null;
        Integer listNumber = null;
        Boolean disabled = null;
        try {
            countryId = userService.getUserModel().getCountryId();
            addressPartId =
Integer.valueOf(String.valueOf(params.get("address_part_id").get(0)));
            listNumber =
Integer.valueOf(String.valueOf(params.get("list_number").get(0)));
            disabled =
Boolean.valueOf(String.valueOf(params.get("disabled").get(0)));
            logger.debug(String.format("params: countryId: %s, addressPartId: %s,
listNumber: %s, disabled: %s", countryId, addressPartId, listNumber, disabled));
            orderService.updateAddressMapping(id, countryId, addressPartId,
listNumber, disabled);
            modelMap.put("success", true);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address", method = RequestMethod.GET)
    public Map<String,?> getAddressFields() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            List<AddressField> addressFields =
orderService.getAddressFieldsLogging();
            if (addressFields != null) {
                logger.debug("addressFields != null");
```

```
                modelMap.put("success", true);
                modelMap.put("data", addressFields);
            }
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/{id}", method = RequestMethod.PUT)
    public Map<String,?> updateAddressField(@PathVariable("id") Integer id,
                                            @RequestBody MultiValueMap<String,
String> params) {
        logger.debug(String.format("id: %s", id));
        Map<String,Object> modelMap = new HashMap<>(3);
        String name = null;
        Boolean disabled = null;
        try {
            name = String.valueOf(String.valueOf(params.get("name").get(0)));
            disabled =
Boolean.valueOf(String.valueOf(params.get("disabled").get(0)));
            logger.debug(String.format("params: mane: %s, disabled: %s", name,
disabled));
            orderService.updateAddressField(id, name, disabled);
            modelMap.put("success", true);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address", method = RequestMethod.POST)
    public Map<String,?> addAddressField(@RequestParam("name") String name) {
        logger.debug(String.format("name: %s", name));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            orderService.addAddressField(name, false);
            modelMap.put("success", true);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/users", method = RequestMethod.GET)
    public Map<String,?> getInitialUserList() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            AbstractUserModel user = userService.getUserModel();
            List<ManagerGridUsers> users;
            if(user.getUserRole() == AbstractUserModel.ROLES.ROLE_SUPER_MANAGER) {
                users = userService.getUsersListForManager(null);
            } else {
                users = userService.getUsersListForManager(user.getCountryId());
```

```
        }
        if (users != null) {
            logger.debug("users!=null");
            modelMap.put("success", true);
            modelMap.put("data", users);
        }
    } catch(Exception e) {
        logger.error(e.getMessage(), e);
        modelMap.put("success", false);
        modelMap.put("error_message", e.getMessage());
    }
    return modelMap;
}
@ResponseBody
@RequestMapping(value="/api/manager/users/{id}", method = RequestMethod.PUT)
public Map<String,?> addUser(@PathVariable Integer id,
                             @RequestBody AbstractUserModel user) {
    logger.debug(String.format("id: %s, user: %s",id, user));
    Map<String, Object> modelMap = new HashMap<>(3);
    GenericPair<Boolean, String> result = null;
    try {
        if(id == -1) {
            logger.debug("id==-1");
            result = userService.addUser(user);
        } else {
            logger.debug("id!=-1");
            result = userService.editUser(user);
        }
        if (!result.getId()) {
            logger.debug(String.format("result.getId(): %s", result.getId()));
            modelMap.put("error", result.getValue());
        }
    } catch(Exception e) {
        logger.error(e.getMessage(), e);
        modelMap.put("error_message", e.getMessage());
    }
    modelMap.put("success", result.getId());
    return modelMap;
}
@ResponseBody
@RequestMapping(value = "/api/manager/users/{id}", method = RequestMethod.GET)
public Map<String, ?> getUser(@PathVariable Integer id) {
    logger.debug(String.format("id: %s", id));
    Map<String, Object> modelMap = new HashMap<>(3);
    try {
        AbstractUserModel user = userService.getUserModel();
        List<ManagerGridUsers> data;
        if(user.getUserRole() == AbstractUserModel.ROLES.ROLE_SUPER_MANAGER) {
            data = userService.getUsersListForManager(null);
        } else {
            data = userService.getUsersListForManager(user.getCountryId());
        }
        logger.debug(String.format("data: %s", data));
        if (data != null) {
            logger.debug("data!=null");
            modelMap.put("success", true);
            modelMap.put("data", data);
        } else {
            logger.debug("data==null");
            modelMap.put("success", false);
```

```
            }
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/checkOrderAddress", method =
RequestMethod.GET)
    public  Map<String,?> checkOrderAddress(@RequestBody CustomerModel customerModel,
                                            @RequestParam("mode") boolean mode){
        logger.debug(String.format("customerModel: %s, mode: %s", customerModel,
mode));
        return DbUtil.checkOrderAddress(orderService,customerModel, mode);
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/customer/orders", method =
RequestMethod.GET)
    public Map<String,?> getOrdersByCustomers(@RequestParam("(bFirstName") String
bFirstName,
                                              @RequestParam("bLastName") String
bLastName,
                                              @RequestParam(value = "bCountry",
required = false) Integer bCountry) {
        bCountry = userService.getUserModel().getCountryId();
        logger.debug(String.format("bFirstName: %s, bLastname: %s, bCountry: %s",
bFirstName, bLastName, bCountry));
        return DbUtil.geCustomerOrders(customerService, bFirstName, bLastName,
bCountry);
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/products", method = RequestMethod.GET)
    public Map<String, ?> getProducts() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            List<BooleanPair> products = shopService.getGlobalProductNames();
            logger.debug(String.format("products: %s", products));
            modelMap.put("success", true);
            modelMap.put("data", products);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/products/{id}", method =
RequestMethod.DELETE)
    public Map<String,Object> deleteOrder(@PathVariable Integer id){
        logger.debug(String.format("id: %s", id));
        //orderService.getProductNamesForOffice();
        Map<String,Object> modelMap = new HashMap<>(3);
        modelMap.put("success", false);
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/stat/productsSold", method =
RequestMethod.GET)
```

```
    public Map<String,?> getStatisticsOnSoldProducts(@RequestParam
@DateTimeFormat(pattern="yyyy-MM-dd") Date dateFrom,
                                                     @RequestParam
@DateTimeFormat(pattern="yyyy-MM-dd") Date dateTo) {
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("countryId: %s, dateFrom: %s, dateTo: %s",
countryId, dateFrom, dateTo));
        Map<String, Object> modelMap = new HashMap<>(3);
        List<ProductStats> productStatsList = null;
        try {
            productStatsList = statisticService.getProductSoldStat(countryId,
dateFrom, dateTo);
            logger.debug(String.format("productStatsList: %s", productStatsList));
            modelMap.put("success", true);
            modelMap.put("data", productStatsList);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/stat/productsBuyout", method =
RequestMethod.GET)
    public Map<String,?> getProductBuyoutStat(@RequestParam
@DateTimeFormat(pattern="yyyy-MM-dd") Date dateFrom,
                                              @RequestParam
@DateTimeFormat(pattern="yyyy-MM-dd") Date dateTo) {
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("countryId: %s, dateFrom: %s, dateTo: %s",
countryId, dateFrom, dateTo));
        Map<String, Object> modelMap = new HashMap<>(3);
        List<ProductStats> productStatsList = null;
        try {
            productStatsList = statisticService.getProductBuyoutStat(countryId,
dateFrom, dateTo);
            logger.debug(String.format("productStatsList: %s", productStatsList));
            modelMap.put("success", true);
            modelMap.put("data", productStatsList);
        } catch(Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/order", method = RequestMethod.GET)
    public Map<String,?> getOrderByName(@RequestParam("name") int name){
        logger.debug(String.format("name: %s", name));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            OrderModel data = orderService.getOrderByOrderNameLogging(name);
            if (data != null){
                modelMap.put("data", data);
                modelMap.put("success", true);
            } else {
                modelMap.put("success", false);
            }
```

```
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/order/{id}", method = RequestMethod.GET)
    public Map<String,?> getOrderById(@PathVariable int id){
        logger.debug(String.format("id: %s", id));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            OrderModel data = orderService.getOrderLogging(id) ;
            if (data != null){
                modelMap.put("data", data);
                modelMap.put("success", true);
            } else {
                modelMap.put("success", false);
            }
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/order/{id}", method = RequestMethod.PUT)
    public Map<String,?> editOrder(@PathVariable int id,
                                   @RequestBody OrderModel orderModel,
                                   @RequestParam("isFormChanged") boolean bForm,
                                   @RequestParam("isProductChanged") boolean
bProduct,
                                   @RequestParam(value = "meta_id", required = false)
Integer metaId) {
        logger.debug(String.format("orderModel: %s, bForm: %s, bProduct: %s",
orderModel, bForm, bProduct));
        Map<String, Object> modelMap = new HashMap<>(3);
        boolean result = false;
        modelMap.put("success", result);
        try {
            CustomerModel customerModel = orderModel.getCustomer();
            if (customerModel!= null && customerModel.getbPhone() != null &&
                    !customerModel.getbPhone().startsWith("+")) {
                logger.debug("customerModel!= null && customerModel.getbPhone() !=
null && !customerModel.getbPhone().startsWith(\"+\")");
                customerModel.setbPhone("+" + orderModel.getCustomer().getbPhone());
            }
            orderService.updateOrder(orderModel, bProduct, bForm,

OrderChangeAnalyzer.analyzeForUpdateFeatures(customerService.getFeaturesPerCustomer(o
rderModel.getCountryId(), orderModel.getCustomer().getId()),
                    orderModel.getFeatures()), metaId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
```

```
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/shipper/orders", method = RequestMethod.GET)
    public Map<String, ?>
getShipperOrderViewRowModelsByOrderNumber(@RequestParam("name") int name) {
        logger.debug(String.format("name: %s", name));
        return DbUtil.getShipperOrdersViewModelByNumber(shipperService, name);
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/status", method = RequestMethod.GET)
    public Map<String, ?> getStatuses() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            List<OrderModel.OrderStatusCounts> res =
orderService.getOrderStatusCounts();
            logger.debug(String.format("res = %s", res));
            modelMap.put("data", res);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/seen", method = RequestMethod.POST)
    public Map<String,?> setNotSeen(@RequestParam(value = "id") Integer id){
        logger.debug(String.format("id: %s", id));
        return DbUtil.setSeen(orderService, userService, webSocketService, id,
false);
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/order/generateNumber", method =
RequestMethod.GET)
    public Map<String,?> generateOrderNumber(){
        return DbUtil.generateOrderNumber(orderService);
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productname", method=RequestMethod.GET)
    public Map<String, ?> getGlobalProduct() {
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", shopService.getGlobalProductNamesLogging());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productname/{id}", method=RequestMethod.PUT)
    public Map<String, ?> editGlobalProduct(@PathVariable("id") int id,
                                            @RequestBody BooleanPair productName) {
        logger.debug(String.format("id: %s, productName: %s", id, productName));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
```

```
            shopService.editGlobalProductName(id, productName.getValue(),
productName.getDisabled());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productname", method=RequestMethod.POST)
    public Map<String, ?> addGlobalProduct(@RequestBody BooleanPair productName) {
        logger.debug(String.format("productName: %s", productName));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            shopService.addGlobalProductName(productName.getValue());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/package", method = RequestMethod.GET)
    public  Map<String,?> getPackages(@RequestParam("productNameId") int
productNameId,
                                      @RequestParam("dosageId") int dosageId){
        logger.debug(String.format("productNameId: %s, dosageId: %s", productNameId,
dosageId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", shopService.getGlobalPackages(productNameId,
dosageId));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/package/{id}", method=RequestMethod.PUT)
    public Map<String, ?> editGlobalPackage(@PathVariable("id") int id,
                                            @RequestBody BooleanPair packageName) {
        logger.debug(String.format("id: %s, packageName: %s", id, packageName));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            shopService.editGlobalPackageName(id, packageName.getValue(),
packageName.getDisabled());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
```

```
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/package", method=RequestMethod.POST)
    public Map<String, ?> addGlobalPackage(@RequestBody BooleanPair packageName) {
        logger.debug(String.format("packageName: %s", packageName));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            shopService.addGlobalPackage(packageName.getValue());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/package/all", method=RequestMethod.GET)
    public Map<String, ?> geGlobalPackages() {
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            List<BooleanPair> res = shopService.getGlobalPackagesLogging();
            modelMap.put("data", res);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/dosage", method = RequestMethod.GET)
    public  Map<String,?> getDosages(@RequestParam("productNameId") int
productNameId){
        logger.debug(String.format("productNameId: %s", productNameId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", orderService.getDosagesForOffice(productNameId));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/dosage/all", method=RequestMethod.GET)
    public Map<String, ?> geGlobalDosages() {
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            List<BooleanPair> res = shopService.getGlobalDosagesLogging();
            modelMap.put("data", res);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
```

```
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/dosage/{id}", method=RequestMethod.PUT)
    public Map<String, ?> editGlobalDosage(@PathVariable("id") int id,
                                           @RequestBody BooleanPair dosage) {
        logger.debug(String.format("id: %s, dosage: %s", id, dosage));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            shopService.editGlobalDosageName(id, dosage.getValue(),
dosage.getDisabled());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/dosage", method=RequestMethod.POST)
    public Map<String, ?> addGlobalDosage(@RequestBody BooleanPair dosage) {
        logger.debug(String.format("dosage: %s", dosage));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            shopService.addGlobalDosage(dosage.getValue());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productlist/{id}/office",
method=RequestMethod.POST)
    public Map<String, ?> addProductListToOffice(@PathVariable("id") Integer
productListId,
                                                 @RequestParam Integer officeId) {
        logger.debug(String.format("productListId: %s, officeId: %s", productListId,
officeId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            orderService.addProductListIdToOffice(productListId, officeId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productlist/{id}/office/{office_id}",
method=RequestMethod.DELETE)
    public Map<String, ?> removeProductListFromOffice(@PathVariable("id") Integer
productListId,
                                                      @PathVariable("office_id")
Integer officeId) {
```

```
        logger.debug(String.format("productListId: %s, officeId: %s", productListId,
officeId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            orderService.removeProductListIdFromOffice(productListId, officeId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productlist/{id}/shipper",
method=RequestMethod.POST)
    public Map<String, ?> addProductListToShipper(@PathVariable("id") Integer
productListId,
                                                  @RequestParam("shipper_id")
List<Integer> shipperId,
                                                  @RequestParam("delivery_id")
Integer deliveryId) {
        logger.debug(String.format("productListId: %s, shipperId: %s, deliveryId:
%s",
                productListId, shipperId, deliveryId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            orderService.addProductListIdToShippers(productListId, shipperId,
deliveryId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/shipper_list", method = RequestMethod.GET)
    public  Map<String,?> getListOfShippers(){
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            AbstractUserModel userModel = userService.getUserModel();
            List<Shipper> data =
shipperService.getListOfShippers(userModel.getCountryId());
            logger.debug(String.format("data: %s", data));
            modelMap.put("data", data);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/shipper_product_list", method =
RequestMethod.GET)
    public  Map<String,?> getListOfShipperProducts(@RequestParam("page") int page,
                                                   @RequestParam("start") int start,
```

```
                                                    @RequestParam("limit") int limit,
                                                    @RequestParam("sort") String sort,
                                                    @RequestParam(value="key",
required=false) String key,
                                                    @RequestParam("shipper_id") int
shipperId){
        logger.debug(String.format("page: %s, start: %s, limit: %s, sort: %s, key:
%s, shipperId: %s",
                page, start, limit, sort, key, shipperId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            JSONArray json =(JSONArray)new JSONParser().parse(sort);
            String column = JsonUtil.getSortingColumnName((JSONObject) json.get(0));
            boolean desc = JsonUtil.getSortingDirection((JSONObject) json.get(0));
            List<ShipperProductSimpleModel> data =
shipperService.getListOfShipperProducts(shipperId,
                    limit, start, column, desc, key);
            int count = shipperService.getListOfShipperProductsCount(shipperId, key);
            modelMap.put("data", data);
            modelMap.put("total", count);
            modelMap.put("success", true);
        } catch (Exception e){
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/templates", method=RequestMethod.GET)
    public Map<String, ?> getTemplates() {
        /*
            GET список получить
            'id', 'idShipper', 'shipper', 'tmpl' , 'disabled'
         */
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", orderService.getTemplates());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/template", method=RequestMethod.POST)
    public Map<String, ?> addTemplate(@RequestParam("shipper_id") Integer shipperId,
                                      @RequestParam("disabled") boolean disabled,
                                      @RequestParam("tmpl") String template) {
        /*
            POST новая строка Например /api/manager/address/template/
            {idShipper, disabled, tmpl }
            tmpl = '{district="так называется поле в  екселе"} {b="второе поле"},
{c="третье поле"}'
         */
        logger.debug(String.format("shipperId: %s, disabled: %s, template: %s",
shipperId, disabled, template));
        Map<String,Object> modelMap = new HashMap<>(3);
```

```
        try {
            orderService.addTemplate(shipperId, disabled, template);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/address/template/{id}",
method=RequestMethod.PUT)
    public Map<String, ?> editTemplate(@PathVariable Integer id,
                                       @RequestBody MultiValueMap<String, String>
params) {
        /*
            PUT
            {idShipper, disabled, tmpl }
            tmpl = '{district="так называется поле в  екселе"} {b="второе поле"},
{c="третье поле"}'
         */
        Map<String,Object> modelMap = new HashMap<>(3);
        Integer shipperId = null;
        Boolean disabled = null;
        String tmpl = null;
        try {
            shipperId =
Integer.valueOf(String.valueOf(params.get("shipper_id").get(0)));
            disabled =
Boolean.valueOf(String.valueOf(params.get("disabled").get(0)));
            tmpl = String.valueOf(params.get("tmpl").get(0));
            logger.debug(String.format("id: %s, shipperId: %s, disabled: %s, tmpl:
%s", id, shipperId, disabled, tmpl));
            orderService.editTemplate(id, shipperId, disabled, tmpl);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/postback/country", method=RequestMethod.GET)
    public Map<String, ?> getPostback() {
        Integer countryId = userService.getUserModel().getCountryId();
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", orderService.getPostBackLogging(countryId));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/postback/country", method=RequestMethod.PUT)
```

```
    public Map<String, ?> editPostback(@RequestBody MultiValueMap<String, String>
params) {
        Map<String,Object> modelMap = new HashMap<>(3);
        Integer countryId = userService.getUserModel().getCountryId();
        String url = null;
        try {
            url = String.valueOf(String.valueOf(params.get("url").get(0)));
            logger.debug(String.format("countryId: %s, url: %s", countryId, url));
            orderService.updatePostBack(countryId, url);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productlist/{id}/shipper/{shipper_id}/
delivery/{delivery_id}", method=RequestMethod.DELETE)
    public Map<String, ?> removeProductListFromShipper(@PathVariable("id") Integer
productListId,
                                                       @PathVariable("shipper_id")
Integer shipperId,
                                                       @PathVariable("delivery_id")
Integer deliveryId) {
        logger.debug(String.format("productListId: %s, shipperId: %s, deliveryId:
%s", productListId, shipperId, deliveryId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            orderService.enableOrDisableProductListIdFromShipper(productListId,
shipperId, deliveryId, true);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productlist", method=RequestMethod.POST)
    public Map<String, ?> addGlobalProductList(@RequestParam Integer productNameId,
                                               @RequestParam Integer packageId,
                                               @RequestParam Integer dosageId) {
        logger.debug(String.format("productNameId: %s, packageId: %s, dosageId: %s",
productNameId, packageId, dosageId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            shopService.addToGlobalProductList(productNameId, packageId, dosageId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/productlist", method=RequestMethod.GET)
```

```
    public Map<String, ?> getProductList(@RequestParam(required = false) Integer
productNameId,
                                         @RequestParam(required = false) Integer
packageId,
                                         @RequestParam(required = false) Integer
dosageId,
                                         @RequestParam(value = "country_id", required
= false) Integer countryId) {
        logger.debug(String.format("productNameId: %s, packageId: %s, dosageId: %s",
productNameId, packageId, dosageId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", shopService.getGlobalProductList(countryId,
                    productNameId, packageId, dosageId));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/scripts/{id}", method=RequestMethod.DELETE)
    public Map<String, ?> deleteScript(@PathVariable("id") Integer scriptId) {
        logger.debug(String.format("scriptId: %s", scriptId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            orderService.deleteScript(scriptId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/scripts/{id}", method=RequestMethod.PUT)
    public Map<String, ?> editScript(@PathVariable("id") Integer scriptId,
                                     @RequestBody MultiValueMap<String, String>
params) {
        Map<String,Object> modelMap = new HashMap<>(3);
        Integer countryId = null;
        Integer productId = null;
        try {
            countryId = userService.getUserModel().getCountryId();
            productId =
Integer.valueOf(String.valueOf(params.get("product_id").get(0)));
            String script = String.valueOf(params.get("script").get(0));
            String scriptName = String.valueOf(params.get("script_name").get(0));
            logger.debug(String.format("scriptId: %s, countryId: %s, productId: %s,
script: %s, scriptName: %s", scriptId, countryId, productId, script, scriptName));
            orderService.editScript(scriptId, countryId, productId, script,
scriptName);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
```

```
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/customer/countries",
method=RequestMethod.GET)
    public Map<String, ?> geCustomerCountries() {
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            List<GenericPair<String, Integer>> res =
customerService.getCountryRootLeafs();
            modelMap.put("data", res);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/customer/countries/states",
method=RequestMethod.GET)
    public Map<String, ?> geCustomerCountryStates(@RequestParam("category") String
category) {
        logger.debug(String.format("category: %s", category));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            List<GenericPair<String, Integer>> res =
customerService.getCountryChildren(category);
            modelMap.put("data", res);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/customer/countries/states/cities",
method=RequestMethod.GET)
    public Map<String, ?> geCustomerCountryStateCities(@RequestParam("category")
String category) {
        logger.debug(String.format("category: %s", category));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            List<GenericPair<String, Integer>> res =
customerService.getStateChildren(category);
            modelMap.put("data", res);
            modelMap.put("success", true);
        } catch (Exception e){
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/scripts", method=RequestMethod.GET)
```

```
    public Map<String, ?> getScripts(@RequestParam(value = "product_id", required =
false) Integer productId) {
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("countryId: %s, productId: %s", countryId,
productId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", orderService.getScripts(countryId, productId));
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/scripts", method=RequestMethod.POST)
    public Map<String, ?> addScript(@RequestParam("script_name") String scriptName,
                                    @RequestParam("product_id") Integer productId,
                                    @RequestParam("script") String script) {
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("scriptName: %s, countryId: %s, productId: %s",
                scriptName, countryId, productId));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            orderService.addScript(scriptName, countryId, productId, script);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/features", method = RequestMethod.GET)
    public Map<String, ?> getFeatureList(){
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("countryId: %s", countryId));
        Map<String, Object> modelMap = new HashMap<>(3);
        List<FeatureModel> features;
        try {
            features = customerService.getFeaturesList(countryId);
            modelMap.put("data", features);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/features/key", method=RequestMethod.POST)
    public Map<String, ?> addFeatureKey(@RequestParam Integer featureId,
                                        @RequestParam String key) {
        logger.debug(String.format("featureId: %s, key: %s", featureId, key));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
```

```
            customerService.addFeatureKey(featureId, key);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/features/key/{id}", method=RequestMethod.PUT)
    public Map<String, ?> editFeatureKey(@PathVariable("id") int keyId,
                                         @RequestBody MultiValueMap<String, String>
keyMap) {
        Map<String,Object> modelMap = new HashMap<>(3);
        String key = String.valueOf(keyMap.get("key").get(0));
        logger.debug(String.format("keyId: %s, key: %s", keyId, key));
        try {
            customerService.editFeatureKey(keyId, key);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/features/key/{id}/enabled",
method=RequestMethod.PUT)
    public Map<String, ?> deleteFeatureKey(@PathVariable("id") Integer keyId,
                                           @RequestBody MultiValueMap<String, String>
params) {
        logger.debug(String.format("keyId: %s, par  : %s", keyId, params));
        Map<String,Object> modelMap = new HashMap<>(3);
        try {
            if(params.get("enabled") == null || params.get("enabled").isEmpty()) {
                modelMap.put("success", false);
                modelMap.put("error_message", "Attribute \"enabled\" is not
present");
            } else {
                boolean enabled = Boolean.parseBoolean(params.get("enabled").get(0));
                customerService.deleteFeatureKey(keyId, enabled);
                modelMap.put("success", true);
            }
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/features", method=RequestMethod.POST)
    public Map<String, ?> addFeature(@RequestParam String label,
                                     @RequestParam String xType) {
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("countryId: %s, label: %s, xType: %s", countryId,
label, xType));
        Map<String,Object> modelMap = new HashMap<>(3);
```

```
        try {
            customerService.addFeature(label, xType, countryId);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/features/{id}", method=RequestMethod.PUT)
    public Map<String, ?> editFeature(@PathVariable("id") int featureId,
                                      @RequestBody MultiValueMap<String, String>
featureName) {
        Map<String,Object> modelMap = new HashMap<>(3);
        String feature = String.valueOf(featureName.get("featureName").get(0));
        String xType = String.valueOf(featureName.get("xType").get(0));
        logger.debug(String.format("feature: %S, xType: %s", feature, xType));
        try {
            customerService.editFeatureName(featureId, feature, xType);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value="/api/manager/features/{id}/visibility",
method=RequestMethod.PUT)
    public Map<String, ?> changeVisibility(@PathVariable("id") int featureId,
                                           @RequestBody MultiValueMap<String, String>
map) {
        Map<String,Object> modelMap = new HashMap<>(3);
        Boolean visibility = Boolean.valueOf(map.get("visibility").get(0));
        logger.debug(String.format("featureId: %s, visibility: %s", featureId,
visibility));
        try {
            customerService.changeVisibility(featureId, visibility);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/heatmap/types", method = RequestMethod.GET)
    public Map<String, ?> getSelectionTypes(@RequestParam("lang") String lang){
        Integer countryId = userService.getUserModel().getCountryId();
        logger.debug(String.format("lang: %s, countryId: %s", lang, countryId));
        Map<String, Object> modelMap = new HashMap<>(3);
        List<SelectionTypeInfo> data = null;
        try{
            data = statisticService.getSelectionTypesInfoByLang(lang, countryId);
            modelMap.put("data", data);
            modelMap.put("success", true);
```

```
        } catch (Exception e){
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/heatmap/types/edit", method =
RequestMethod.PUT)
    public Map<String, ?> editSelectionType(@RequestBody MultiValueMap<String,
String> params){
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            SelectionTypeInfo newInfo = new SelectionTypeInfo();
            if(params.get("typeKey") != null && !
params.get("typeKey").get(0).isEmpty()) {
                newInfo.setTypeKey((Integer.parseInt(params.get("typeKey").get(0))));
            }
            newInfo.setName(params.get("name").get(0));
            newInfo.setLang(params.get("lang").get(0));
            newInfo.setCountryId(userService.getUserModel().getCountryId());
            newInfo.setDescription(params.get("description").get(0));
            logger.debug(String.format("newInfo: %s", newInfo));
            statisticService.updateSelectionTypeInfo(newInfo);
            modelMap.put("success", true);
        } catch (Exception e){
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/heatmap/types/delete/{id}", method =
RequestMethod.DELETE)
    public Map<String, ?> deleteSelectionTypeInfo(@PathVariable("id") int id){
        logger.debug(String.format("id: %s", id));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            statisticService.deleteSelectionTypeInfo(id);
            modelMap.put("success", true);
        } catch (Exception e){
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/users/checkVoipLoginAvailable", method =
RequestMethod.GET)
    public Map<String, ?> checkVoipExistence(@RequestParam("voip_login") String
voipLogin,
                                             @RequestParam("office_id") int officeId)
{
        logger.debug(String.format("voipLogin: %s, officeId: %s", voipLogin,
officeId));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
```

```
            GenericPair<Boolean, String> result =
userService.checkVoipExistenceWithLogin(voipLogin, officeId);
            modelMap.put("login_available", result.getId());
            modelMap.put("error", result.getValue());
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/criteria/addCriteria", method =
RequestMethod.POST)
    public Map<String, ?> addQualityCheckCriteria(@RequestParam("label") String
label,
                                                  @RequestParam("type") String type){
        logger.debug(String.format("label: %s, type: %s", label, type));
        int countryId = userService.getUserModel().getCountryId();
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            qualityService.addQualityCheckCriteria(countryId, label, type);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/criteria/{id}", method = RequestMethod.PUT)
    public Map<String, ?> editQualityCheckCriteria(@PathVariable("id") int id,
@RequestBody MultiValueMap<String, String> params){
        Map<String, Object> modelMap = new HashMap<>(3);
        String newLabel = params.get("label").get(0);
        String newType = params.get("type").get(0);
        logger.debug(String.format("id: %s, newLabel: %s, newType: %s", id, newLabel,
newType));
        try {
            qualityService.editQualityCheckCriteria(id, newType, newLabel);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/criteria/keys", method =
RequestMethod.POST)
    public Map<String, ?> addQualityCheckCriteriaKey(@RequestParam("criteria_id") int
criteriaId,
                                                     @RequestParam("key") String key)
{
        logger.debug(String.format("criteriaId: %s, key: %s", criteriaId, key));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
```

```
            qualityService.addQualityCheckCriteriaKey(criteriaId, key);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/criteria/key/{id}", method =
RequestMethod.PUT)
    public Map<String, ?> editQualityCheckCriteriaKey(@PathVariable("id") int
criteriaKeyId,
                                                      @RequestBody
MultiValueMap<String, String> params){
        Map<String, Object> modelMap = new HashMap<>(3);
        String newKey = params.get("key").get(0);
        logger.debug(String.format("criteriaKeyId: %s, newKey: %s", criteriaKeyId,
newKey));
        try {
            qualityService.editQualityCheckCriteriaKey(criteriaKeyId, newKey);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/criteria/key/{id}/enabled", method =
RequestMethod.PUT)
    public Map<String, ?> deleteQualityCheckCriteriaKey(@PathVariable("id") int id,
                                                        @RequestBody
MultiValueMap<String, String> params){
        logger.debug(String.format("id: %s", id));
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if(params.get("enabled") == null || params.get("enabled").isEmpty()) {
                modelMap.put("success", false);
                modelMap.put("error_message", "Attribute \"enabled\" is not
present");
            } else {
                boolean enabled = Boolean.parseBoolean(params.get("enabled").get(0));
                qualityService.enabledQualityCheckCriteriaKey(id, enabled);
                modelMap.put("success", true);
            }
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/criteria/{id}/visibility", method =
RequestMethod.PUT)
    public Map<String, ?> changeEnabled(@PathVariable("id") int criteriaId,
```

```
                                        @RequestBody MultiValueMap<String, String>
params){
        Map<String, Object> modelMap = new HashMap<>(3);
        Boolean enabled = Boolean.valueOf(params.get("enabled").get(0));
        logger.debug(String.format("criteriaId: %s, enabled: %s", criteriaId,
enabled));
        try {
            qualityService.changeEnabled(criteriaId, enabled);
            modelMap.put("success", true);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/workTime/time", method = RequestMethod.GET)
    public Map<String, ?> getTimeIntervals(@RequestParam("start_period")
@DateTimeFormat(pattern = "yyyy-MM-dd") Date from,
                                           @RequestParam("end_period")
@DateTimeFormat(pattern = "yyyy-MM-dd") Date to,
                                           @RequestParam("operator_id") List<Integer>
operatorIds){
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            modelMap.put("data", statisticService.getTimeIntervalsByPeriod(from, to,
operatorIds));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/workTime/amount", method =
RequestMethod.GET)
    public Map<String, ?> getAmountEvents(@RequestParam("start_period")
@DateTimeFormat(pattern = "yyyy-MM-dd") Date from,
                                          @RequestParam("end_period")
@DateTimeFormat(pattern = "yyyy-MM-dd") Date to,
                                          @RequestParam("operator_id") List<Integer>
operatorIds){
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            modelMap.put("data", statisticService.getAmountEventsByPeriod(from, to,
operatorIds));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/operators", method = RequestMethod.GET)
    public Map<String, ?> getOperators() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
```

```
            modelMap.put("data",
userService.getOperatorsPerCountry(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/productsList", method = RequestMethod.GET)
    public Map<String, ?> getProductLists() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
userService.getProductsPerCountry(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/promo", method = RequestMethod.GET)
    public Map<String, ?> getAllPromoByCountry(){
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            modelMap.put("data",
orderService.getPromoByCountry(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/promo", method = RequestMethod.POST)
    public Map<String, ?> addPromo(@RequestBody Promo promo){
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            orderService.addPromo(promo, userService.getUserModel().getCountryId());
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/promo", method = RequestMethod.PUT)
    public Map<String, ?> updatePromo(@RequestBody Promo promo){
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            orderService.updatePromo(promo);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
```

```
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/promo/{id}", method = RequestMethod.PUT)
    public Map<String, ?> changeEnabledPromo(@PathVariable("id") int promoId,
                                             @RequestBody MultiValueMap<String,
String> params){
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            boolean enabled;
            if(params.get("enabled") != null) {
                enabled = Boolean.parseBoolean(params.get("enabled").get(0));
                orderService.changeEnabledPromo(promoId, enabled);
                modelMap.put("success", true);
            } else {
                modelMap.put("success", false);
                modelMap.put("error_message", "Parameter 'enabled' is not found");
            }
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group", method = RequestMethod.GET)
    public Map<String, ?> getOperatorGroups() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            modelMap.put("data",
userService.getOperatorGroupsPerCountry(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group", method = RequestMethod.POST)
    public Map<String, ?> addOperatorGroup(@RequestBody OperatorGroup group) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            userService.addOperatorGroup(group,
userService.getUserModel().getCountryId());
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group", method = RequestMethod.PUT)
    public Map<String, ?> updateOperatorGroup(@RequestBody OperatorGroup group) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try{
            userService.updateOperatorGroup(group);
            modelMap.put("success", true);
```

```
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{id}", method = RequestMethod.PUT)
    public Map<String, ?> enableOperatorGroup(@PathVariable("id") int groupId,
                                              @RequestBody MultiValueMap<String,
String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            boolean enabled;
            if(params.get("enabled") != null) {
                enabled = Boolean.parseBoolean(params.get("enabled").get(0));
                userService.changeEnabledOperatorGroup(groupId, enabled);
                modelMap.put("success", true);
            } else {
                modelMap.put("success", false);
                modelMap.put("error_message", "Parameter 'enabled' is not found");
            }
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/users", method =
RequestMethod.GET)
    public Map<String, ?> getOperatorsFromGroup(@PathVariable("group_id") int
groupId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", userService.getOperatorsFromGroup(groupId));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/user/{operator_id}",
method = RequestMethod.PUT)
    public Map<String, ?> addOperatorToGroup(@PathVariable("group_id") int groupId,
                                             @PathVariable("operator_id") int
operatorId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.addOperatorToGroup(groupId, operatorId,
userService.getUserModel().getCountryId());
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
```

```
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/user/{operator_id}",
method = RequestMethod.DELETE)
    public Map<String, ?> removeOperatorFromGroup(@PathVariable("group_id") int
groupId,
                                                  @PathVariable("operator_id") int
operatorId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.deleteOperatorFromGroup(groupId, operatorId,
userService.getUserModel().getCountryId());
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/products", method =
RequestMethod.GET)
    public Map<String, ?> getProductsFromGroup(@PathVariable("group_id") int groupId)
{
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", userService.getProductsFromGroup(groupId));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/product/{product_id}",
method = RequestMethod.PUT)
    public Map<String, ?> addProductToOperatorGroup(@PathVariable("group_id") int
groupId,
                                                    @PathVariable("product_id") int
productId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.addProductToGroup(groupId, productId);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/product/{product_id}",
method = RequestMethod.DELETE)
    public Map<String, ?> removeProductFromOperatorGroup(@PathVariable("group_id")
int groupId,
                                                         @PathVariable("product_id")
int productId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.deleteProductFromGroup(groupId, productId);
```

```
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/group/{group_id}/promo", method =
RequestMethod.GET)
    public Map<String, ?> getPromoByOperatorGroup(@PathVariable("group_id") int
groupId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", orderService.getPromoByGroup(groupId));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog/template", method =
RequestMethod.POST)
    public Map<String, ?> updateLogTemplate(@RequestParam("event") String event,
                                            @RequestParam("lang") String lang,
                                            @RequestParam("template") String
template) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            int countryId = userService.getUserModel().getCountryId();
            loggingService.updateTemplate(EventType.valueOf(event),
lang.toUpperCase(), template, countryId);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog/templates", method =
RequestMethod.GET)
    public Map<String, ?> getUserlogTemplates() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {

modelMap.put("data",loggingService.getTemplates(userService.getUserModel().getCountry
Id()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog/events", method =
RequestMethod.GET)
```

```
    public Map<String, ?> getUserlogEvents() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data", EventType.values());
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog/settings", method =
RequestMethod.GET)
    public Map<String, ?> getUserlogSettings() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
loggingService.getSettingsForCountry(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog/settings", method =
RequestMethod.POST)
    public Map<String, ?> updateUserlogSettings(@RequestBody Map<EventType, Boolean>
settings) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {

loggingService.setSettingsForCountry(userService.getUserModel().getCountryId(),
settings);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog", method = RequestMethod.GET)
    public Map<String, ?> getUserlog(@RequestParam("lang") String lang,
                                     @RequestParam(value = "start_date", required =
false) @DateTimeFormat(pattern = "yyyy-MM-dd") Date from,
                                     @RequestParam(value = "end_date", required =
false) @DateTimeFormat(pattern = "yyyy-MM-dd") Date to,
                                     @RequestParam(value = "user_id", required =
false) Integer userId,
                                     @RequestParam("page") int page,
                                     @RequestParam("limit") int limit,
                                     @RequestParam("sort") String sort) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            JSONArray json =(JSONArray)new JSONParser().parse(sort);
            String column = JsonUtil.getSortingColumnName((JSONObject)json.get(0));
            boolean desc = JsonUtil.getSortingDirection((JSONObject) json.get(0));
```

```
            modelMap.put("data",
loggingService.getUserLog(userService.getUserModel().getCountryId(),
                    lang.toUpperCase(), from, to, userId, page, limit, column,
desc));
            modelMap.put("total",
loggingService.getUserLogCount(userService.getUserModel().getCountryId(),
                    lang.toUpperCase(), from, to, userId));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/userlog/description", method =
RequestMethod.GET)
    public Map<String, ?> getUserlog(@RequestParam("event") String event) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
loggingService.getDescriptionByEvent(userService.getUserModel().getCountryId(),
EventType.valueOf(event)));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/delivery", method = RequestMethod.POST)
    public Map<String, ?> addDeliveryOption(@RequestParam("name") String name,
                                            @RequestParam("days") int dayPeriod,
                                            @RequestParam("cost") BigDecimal cost,
                                            @RequestParam("enabled") boolean enabled)
{
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            shipperService.addDeliveryOption(name, dayPeriod, cost, enabled,
userService.getUserModel().getCountryId());
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/delivery", method = RequestMethod.PUT)
    public Map<String, ?> updateDeliveryOption(@RequestBody DeliveryModel
deliveryModel) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            shipperService.updateDeliveryOption(deliveryModel);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
```

```
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/delivery/enabled/{id}", method =
RequestMethod.PUT)
    public Map<String, ?> changeEnabledDeliveryOption(@PathVariable("id") int
deliveryId,
                                                      @RequestBody
MultiValueMap<String, String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        if(params.get("enabled") == null || params.get("enabled").isEmpty()) {
            modelMap.put("success", false);
            modelMap.put("error_message", "One or more parameters are not defined");
        }
        try {
            boolean enabled = Boolean.parseBoolean(params.get("enabled").get(0));
            shipperService.changeEnabledDeliveryOption(deliveryId, enabled);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/users/breaks/amount", method =
RequestMethod.GET)
    public Map<String, ?> getMaxBreaks() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
userService.getMaxUserBreaks(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/users/breaks/amount", method =
RequestMethod.POST)
    public Map<String, ?> setMaxBreaks(@RequestParam("max_amount") Integer maxAmount)
{
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            userService.setMaxUserBreaks(userService.getUserModel().getCountryId(),
maxAmount);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/templates/description", method =
RequestMethod.GET)
    public Map<String, ?> getMessageTemplateDescription() {
        Map<String, Object> modelMap = new HashMap<>(3);
```

```
        try {
            modelMap.put("data",
sendingService.getTemplateDescription(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/templates", method =
RequestMethod.GET)
    public Map<String, ?> getMessageTemplates() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
sendingService.getMessageTemplatesLogging(userService.getUserModel().getCountryId()))
;
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/template", method =
RequestMethod.POST)
    public Map<String, ?> addMessageTemplate(@RequestBody MessageTemplate template) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {

sendingService.addMessageTemplate(userService.getUserModel().getCountryId(),
template);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/template", method =
RequestMethod.PUT)
    public Map<String, ?> updateMessageTemplate(@RequestBody MessageTemplate
template) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            sendingService.updateMessageTemplate(template);
            modelMap.put("success", true);
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/template", method =
RequestMethod.DELETE)
```

```
    public Map<String, ?> deleteMessageTemplate(@RequestBody MultiValueMap<String,
String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if(params.get("template_id") == null ||
params.get("template_id").isEmpty()) {
                modelMap.put("success", false);
                modelMap.put("error_message", "Response code: 400. Required parameter
'template_id' are not defined");
            } else {

sendingService.deleteMessageTemplate(Integer.parseInt(params.get("template_id").get(0
)));
                modelMap.put("success", true);
            }
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/phrase", method =
RequestMethod.POST)
    public Map<String, ?> addPreparedPhrases(@RequestBody MultiValueMap<String,
String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            String phrase = null;
            if(params.get("value") != null && !params.get("value").isEmpty()){
                phrase = params.get("value").get(0);

sendingService.addPreparedPhrase(userService.getUserModel().getCountryId(), phrase);
                modelMap.put("success", true);
            } else {
                modelMap.put("success", false);
                modelMap.put("error_message", "Response code: 400. Required parameter
'value' are not defined.");
            }
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/phrase", method =
RequestMethod.PUT)
    public Map<String, ?> updatePreparedPhrase(@RequestBody MultiValueMap<String,
String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            Integer phraseId = null;
            String newPhrase = null;
            if(params.get("id") != null && params.get("value") != null &&
                    !params.get("id").isEmpty() && !params.get("value").isEmpty()){
                newPhrase = params.get("value").get(0);
                phraseId = Integer.parseInt(params.get("id").get(0));
                sendingService.updatePreparedPhrase(phraseId, newPhrase);
                modelMap.put("success", true);
```

```
            } else {
                modelMap.put("success", false);
                modelMap.put("error_message", "Response code: 400. One or more
parameters are not defined.");
            }
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/phrase", method =
RequestMethod.DELETE)
    public Map<String, ?> deletePreparedPhrase(@RequestBody MultiValueMap<String,
String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if (params.get("id") == null || params.get("id").isEmpty()) {
                modelMap.put("success", false);
                modelMap.put("error_message", "Response code: 400. Required parameter
'id' are not defined");
            } else {

sendingService.deletePreparedPhrase(Integer.parseInt(params.get("id").get(0)));
                modelMap.put("success", true);
            }
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/sending/byPhoneNumbers", method =
RequestMethod.POST)
    public Map<String, ?> manualSendingByPhoneNumbers(@RequestBody
MultiValueMap<String, String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if(params.get("numbers") == null || params.get("message") == null ||
params.get("message").isEmpty() ||
                    params.get("sending_date") == null ||
params.get("sending_date").isEmpty() ||
                    params.get("type") == null || params.get("type").isEmpty() ||
                    params.get("office_id") == null ||
params.get("office_id").isEmpty()) {
                modelMap.put("success", false);
                modelMap.put("error_message", "Response code: 400. One or more
required parameters are not defined");
            } else {
                AbstractUserModel user = userService.getUserModel();
                modelMap.put("uuid", sendingService.manualSending(
                        user.getCountryId(),
                        user.getId(),
                        Integer.parseInt(params.get("office_id").get(0)),
                        params.get("numbers"),
                        params.get("message").get(0),
```

```
Date.from(LocalDateTime.parse(params.get("sending_date").get(0)).atZone(ZoneId.system
Default()).toInstant()),
                        Message.Type.valueOf(params.get("type").get(0)),
                        ManualSendingSupplier.TypeOfSending.BY_PHONE_NUMBERS));
                modelMap.put("success", true);
            }
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/sending/byOrderNumbers", method =
RequestMethod.POST)
    public Map<String, ?> manualSendingByOrderNumbers(@RequestBody
MultiValueMap<String, String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if(params.get("numbers") == null
                    || params.get("message") == null ||
params.get("message").isEmpty() ||
                    params.get("sending_date") == null ||
params.get("sending_date").isEmpty() ||
                    params.get("type") == null || params.get("type").isEmpty() ||
                    params.get("office_id") == null ||
params.get("office_id").isEmpty()) {
                modelMap.put("success", false);
                modelMap.put("error_message", "Response code: 400. One or more
required parameters are not defined");
            } else {
                AbstractUserModel user = userService.getUserModel();
                modelMap.put("uuid", sendingService.manualSending(
                        user.getCountryId(),
                        user.getId(),
                        Integer.parseInt(params.get("office_id").get(0)),
                        params.get("numbers"),
                        params.get("message").get(0),

Date.from(LocalDateTime.parse(params.get("sending_date").get(0)).atZone(ZoneId.system
Default()).toInstant()),
                        Message.Type.valueOf(params.get("type").get(0)),
                        ManualSendingSupplier.TypeOfSending.BY_ORDER_NUMBERS));
                modelMap.put("success", true);
            }
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages", method = RequestMethod.GET)
    public Map<String, ?> getMessages(@RequestParam("start_period")
@DateTimeFormat(pattern="yyyy-MM-dd") Date from,
                                      @RequestParam("end_period")
@DateTimeFormat(pattern="yyyy-MM-dd") Date to,
                                      @RequestParam int page,
                                      @RequestParam int limit,
```

```
                                        @RequestParam String sort,
                                        @RequestParam(required = false) String phone,
                                        @RequestParam(value = "order_number", required
= false) Integer orderNumber,
                                        @RequestParam(value = "old_status", required =
false) List<OrderStatus> oldStatus,
                                        @RequestParam(value = "new_status", required =
false) List<OrderStatus> newStatus) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            JSONArray json =(JSONArray)new JSONParser().parse(sort);
            String column = JsonUtil.getSortingColumnName((JSONObject)json.get(0));
            boolean desc = JsonUtil.getSortingDirection((JSONObject) json.get(0));
            modelMap.put("data",
sendingService.getMessages(userService.getUserModel().getCountryId(), from, to, page,
                    limit, column, desc, orderNumber, phone, oldStatus, newStatus));
            modelMap.put("total",
sendingService.getMessagesCount(userService.getUserModel().getCountryId(), from, to,
                    orderNumber, phone, oldStatus, newStatus));
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/web-hook/sms", method =
RequestMethod.POST)
    public Map<String, ?> smsWebHook(@RequestBody Report report) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            MessageFromReport message = null;
            if(report.getMessages() != null && report.getMessages().size() == 1) {
                message = report.getMessages().get(0);
                sendingService.markMessageAsSentAndDelivered(report.getJob_uuid(),
message.isSent(), message.isDelivered());
                modelMap.put("success", true);
            }
        } catch (Exception e){
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/sms-token", method =
RequestMethod.GET)
    public Map<String, ?> getSmsTokens() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
sendingService.getSmsTokens(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
```

```
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/sms-token", method =
RequestMethod.POST)
    public Map<String, ?> addSmsToken(@RequestBody GenericTrio<Integer, Integer,
String> token) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            sendingService.addSmsToken(userService.getUserModel().getCountryId(),
token.getKey(), token.getValue());
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/sms-token", method =
RequestMethod.PUT)
    public Map<String, ?> updateSmsTokens(@RequestBody GenericTrio<Integer, Integer,
String> token) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            sendingService.updateSmsToken(token.getId(), token.getKey(),
token.getValue());
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/messages/sms-token", method =
RequestMethod.DELETE)
    public Map<String, ?> deleteSmsTokens(@RequestBody GenericTrio<Integer, Integer,
String> token) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            sendingService.deleteSmsToken(token.getId());
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/orders/findByTrackingInfo", method =
RequestMethod.GET)
    public Map<String, ?> getOrdersByTracking(@RequestParam(value = "click_id",
required = false) String clickId,
                                              @RequestParam(value = "track_id",
required = false) String trackId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            int countryId = userService.getUserModel().getCountryId();
            List<OrderModel> orderModels =
orderService.getOrdersByTrackingInfo(countryId, clickId, trackId);
```

```
            modelMap.put("data",
TrackingTableViewModel.buildTrackingTableViewModels(orderModels));
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/orders/emptyQueueNotification/settings",
method = RequestMethod.GET)
    public Map<String, ?> getEmptyQueueNotificationSettings() {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
orderService.getEmptyQueueNotifSettings(userService.getUserModel().getCountryId()));
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/orders/emptyQueueNotification/settings",
method = RequestMethod.PUT)
    public Map<String, ?> updateEmptyQueueNotificationSettings(@RequestBody
MultiValueMap<String, String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if(!params.containsKey("amount_orders") || !
params.containsKey("pause_duration")) {
                modelMap.put("success", false);
                modelMap.put("error_message", "One or more required parameters are
not defined");
                return modelMap;
            }
            int countryId = userService.getUserModel().getCountryId();
            int amountOrders = Integer.parseInt(params.get("amount_orders").get(0));
            int pauseDuration =
Integer.parseInt(params.get("pause_duration").get(0));
            orderService.updateEmptyQueueNotifSettings(countryId, amountOrders,
pauseDuration);
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/groups/products/exceptGroup", method =
RequestMethod.GET)
    public Map<String, ?> getProductsExceptGroup(@RequestParam("group_id") int
groupId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
```

```
            modelMap.put("data",
userService.getProductsExceptGroup(userService.getUserModel().getCountryId(),
groupId));
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/groups/users/exceptGroup", method =
RequestMethod.GET)
    public Map<String, ?> getOperatorsExceptGroup(@RequestParam("group_id") int
groupId) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            modelMap.put("data",
userService.getOperatorsExceptGroup(userService.getUserModel().getCountryId(),
groupId));
            modelMap.put("success", true);
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
    @ResponseBody
    @RequestMapping(value = "/api/manager/orders/number", method = RequestMethod.PUT)
    public Map<String, ?> updateOrderNumber(@RequestBody MultiValueMap<String,
String> params) {
        Map<String, Object> modelMap = new HashMap<>(3);
        try {
            if(params.containsKey("order_id") && params.containsKey("order_number")
&&
                !params.get("order_id").isEmpty() && !
params.get("order_number").isEmpty()) {
                int orderId = Integer.parseInt(params.get("order_id").get(0));
                int orderNumber =
Integer.parseInt(params.get("order_number").get(0));

orderService.updateOrderNumber(userService.getUserModel().getCountryId(), orderId,
orderNumber);
                modelMap.put("success", true);
            } else {
                modelMap.put("success", false);
                modelMap.put("error_message", "One or more parameters are not
defined");
            }
        } catch (Exception e) {
            modelMap.put("success", false);
            modelMap.put("error_message", e.getMessage());
        }
        return modelMap;
    }
}
```